A Second Course in Logic

Christopher Gauker

Version of March 2010

Introduction

These notes are intended for readers who have already had a solid course in contemporary symbolic logic. I assume that readers know how to interpret (informally) the symbols of symbolic logic and have learned how to do proofs in a natural deduction system such as that found in Jon Barwise and John Etchemendy's textbook, *Language, Proof and Logic*. The course picks up at the point where students need to learn a precise definition of truth in a structure and learn to use it to demonstrate the validity and invalidity of arguments.

At the University of Cincinnati, the Philosophy Department teaches a two-quarter logic course. (Quarters are 11-week terms.) In the first eight weeks of the first quarter, the students study material in the first 13 chapters of the Barwise and Etchemendy's textbook. (We skip some sections.) These notes pertain to the last two weeks of the first quarter and all of the second quarter.

If you, dear reader, discover any errors, typographical or worse, I would be grateful for your feedback. Contact me at: christopher.gauker@uc.edu

Outline of course

Here, in outline, is what we will do in this course:

I. Define first-order validity precisely. To do that we need definitions of structures, variable assignments, satisfaction by a variable assignment in a structure, and truth in a structure.

II. Prove soundness and completeness. That is, we prove that the class of first-order valid arguments exactly coincides with the class of arguments whose conclusions can be derived from their premises using our inference rules.

    Dividends that we will reap from this include:

    A. The Löwenheim-Skolem Theorem. If a theory (a set of sentences) has a model (a structure in which all of the sentences in the set are true), then it has a model with a denumerable domain. (So even a theory of the real numbers will be interpretable as true in a model whose domain contains just the positive integers.)

B. The Compactness Theorem. If every finite subset of a set of sentences is consistent, then the whole set is consistent. This does not look very exciting, but it will be exciting to discover, later on, that second-order logic is *not* compact.

III. Prove Gödel's First Incompleteness Theorem. What this says is that the truths of arithmetic are not axiomatizable. That is, there is no decidable, consistent set of sentences such that all of the truths of arithmetic are first-order consequences of the sentences in that set. Not even if the set of axioms is infinite (so long as it is decidable).

Some dividends that we will reap along the way include:

A. We will acquire the concepts of decidability and enumerability and will learn how they can be defined in a precise way in terms of the formulas of arithmetic.

B. We will prove Tarski's undefinability theorem, which says that no bivalent language can contain its own truth predicate.

Actually, we will prove the Gödel theorem in several different ways and in two importantly different versions.

IV. Undecidability of first-order logic. First-order logic is axiomatizable. (For instance, our Fitch inference rules constitute such an axiomatization.) However, it is not decidable. That is, there is no algorithm that will take any argument and tell us whether or not that argument is first-order valid. This will follow pretty quickly from some results that we will have proved on the way to the second version of Gödel's First Incompleteness Theorem.

V. Gödel's Second Incompleteness Theorem. We will make short schrift of this by starting with some big assumptions that everyone believes but no one bothers to prove.

VI. Second-order logic. Second-order logic is just like first-order logic, except that we will have variables that stand in predicate position and we will have quantifiers that bind those variables. The important fact about second-order logic that we will learn is that, like arithmetic, it is not axiomatizable. We can define logical validity for second-order logic, but then we cannot have a set of axioms and inference rules that allow us to give proofs for all and only the second-order valid arguments.

VII. Modal logic. We will define logical validity for a language containing modal operators, such as "necessarily" and "possibly". This gets a bit tricky and controversial when we try to add quantifiers as well. This last unit will belong to what is called "philosophical logic", whereas everything prior will belong to (elementary) mathematical logic.

Sources

Authors of logic books are not very good about telling us where they learned what they know. Maybe they would like us to believe that they made it up themselves! But I certainly did not make this stuff up myself; so here I will list the books from which I have learned the things that I explain in these notes:

Jon Barwise and John Etchemendy, *Language, Proof and Logic*, CSLI (1999-2002). I have taught elementary logic from countless textbooks over the years; this is clearly the best. The computer programs that come with the textbook are an excellent teaching tool. My proof of the completeness theorem (i.e., the proof of the completeness of the Fitch-style natural deduction system with respect to the definition of logical validity) comes from chapters 17 and 19 of this book. However, I have filled in many details that they omit. Strangely, they do not prove a soundness theorem at all (contenting themselves with only the most hand-waving sketch); so I have had to construct that from scratch. With axiomatic proof theories, the proof of soundness is quite trivial; it's not quite so trivial for natural deduction systems.

Raymond Smullyan, *Gödel's Incompleteness Proofs*, Oxford University Press (1992). This is a brilliant book. Although I had previously taught the proofs of Gödel's theorems in Enderton and Boolos & Jeffrey, I probably never really understood Gödel's theorems until I read this book. Smullyan avoids a great deal of complexity by defining recursively enumerable sets and relations as Σ_1 sets and relations. (I will explain these concepts below.) The proofs of incompleteness that I have given below follow certain threads that I have picked out of this book. In one place I refer the reader to this book for a detail that I do not myself provide (but which the reader will probably be content to grant without proof). Strangely, Smullyan goes almost right up to, but then does not actually prove the very general version of Gödel's Theorem proved in Enderton and Boolos & Jeffrey, which I prove in Lesson 10 below. I don't understand why Oxford cannot bring out an inexpensive paperback edition of this book. Maybe it's not Smullyan's fault, but somebody did a very bad job with the index and references.

George Boolos and Richard Jeffrey, *Computability and Logic*, 2nd edition (or 3rd edition), Cambridge University Press. My proof of the more general version of Gödel's Theorem borrows some final steps from this book. Also, my proof of the undecidability of first-order logic is based on the one in this book. This book is widely used in Philosophy, but it's a little strange since it starts out with chapters on computability and then does everything else in terms of that. There is a fourth edition, which adds another author, John Burgess. That one looks significantly different (the type has been completely re-set), but I have never read it.

Herbert Enderton, *Introduction to Mathematical Logic*, Academic Press (1972). This was once the standard textbook, and maybe it still is if Boolos and Jeffrey has not eclipsed it. The proofs are sometimes sketchy, and sometimes key steps do not stand out clearly. The proof of the representability (in a subtheory of number theory) of every recursive function is very hard to penetrate. In these notes I have relied on this text only in the presentation of second order logic.

G. E. Hughes and M. J. Cresswell, *A New Introduction to Modal Logic,* Routledge (1996). I can't say that I learned what I know about modal logic from this book, since I did not even read it until 2004. Nonetheless, this book does contain all of the essential information (as well as much more than the essentials).

Stewart Shapiro, *Foundations without Foundationalism: A Case for Second-order Logic*, Oxford (1991).

Lesson 1: First-order Validity

Throughout, I will assume that we are dealing with a particular first-order language $\mathcal{L}$. I will assume that the reader knows the usual formation rules for a first-order language.

Recall the definition of tautological validity:

An argument is **tautologically valid** if and only if for each **assignment of truth values** to the noncompound components, if the premises are true on that assignment, then the conclusion is true on that assignment too.

The definition of first-order validity will look similar:

An argument is **first-order valid** if and only if for each **structure** of the language, if the premises are true in that structure, then the conclusion is true in that structure too.

So, to understand this definition, we need to know: What is a **structure**?

Very approximately: A structure is a thing that will assign a definite "meaning" or, more accurately, "reference", to each basic vocabulary item, that is, to each name and each predicate. Here, for example, are two partial specifications of structures.

Structure One:
Interprets "a" as standing for a big tetrahedron in the lower left-hand corner of a certain grid.
Interprets "Cube" as standing for the set of cubes on the grid.
Interprets "Larger" as standing for the set of pairs, such that the first member larger than the second.

Structure Two:
Interprets "a" as standing for a small dodecahron in the upper right-hand corner.
Interprets "Cube" as standing for the dodecs on the left-hand side and the large-blocks on the right-hand side.
Interprets "Larger" as standing for the set of pairs such that the first member is a tetrahedron and the second member is a cube.

Why this will be a little bit complicated

Now we want to explain how the truth value of a sentence of a first-order language depends on the interpretation of its components. In propositional logic, this is straightforward because the truth value of a compound sentence is determined by the truth values of its components.

> *Example:* If "Cube(a)" is **False**, and "Large(a)" is **True**, then we know that the truth value of "Cube(a) → Large(a)" is **True**.

In first-order logic (quantifier logic), the truth value of a complex formula is not determined by the truth values of its components, because the components are not always sentences.

> *Example:* "∀x∃yLikes(x, y)" has "Likes(x, y)" as a component, but this is neither true nor false, because it is not even a sentence.

The solution will be to proceed in three steps:

In association with a structure, we will assign "temporary" meanings to variables, via *variable assignments*.

We will first define "Variable assignment *g satisfies* formula **P** in structure **M**".

Then, in terms of satisfaction by a variable assignment we will define "Sentence **P** is true in **M**".

Notational conventions:

When I want to talk about an actual formula or vocabulary item of the language $\mathcal{L}$, I will put it in quotation marks:

> "Cube(a)"
>
> "∃xLarger(y, x)"
>
> "¬"

But sometimes (even very soon) I will omit the quotation marks just to avoid clutter.

When I want to talk about all vocabulary items of a certain kind, or all formulas of a certain form, I will use bold-face sans serif:

> If **n** is a name of $\mathcal{L}$, then ….
>
> For any formula of the form **(P ∨ Q)**, if either…

Note especially: Some formulas of the form **∃vP**:

> ∃xCube(x)
> ∃y(Cube(y) ∧ ¬Small(y))
> ∃x(Cube(x) ∧ ∀y(x≠y → Larger(x, y)))
> ∃x∃yTet(y)

Similarly: Some formulas of the form **∀vQ**:

> ∀xCube(x)
> ∀x¬∃yLarger(x, y)
> ∀y(Cube(y) → ∃x(Tet(x) ∧ Adjoins(y, x)))

Note, though, that in subsequent lessons, I will cease to use boldface and just leave it to context to determine whether I am talking about a sentence of $\mathcal{L}$ or am using schematic letters.

Sets

I will assume that the reader has an intuitive understanding of the concept of a set and understands that the membership of a finite set can be specified by writing names of the members of the set between curly brackets. Also, the order in which we list the members does not affect the identity of the set. For example:

> The set consisting of UC philosophy professors =
>
> {Gauker, Martin, Robinson, Polger, Skipper, Faaborg, Jost, Richardson, Maglo, Carbonell, Allen}
>
> =
>
> {Robinson, Gauker, Martin, Carbonell, Polger, Skipper, Allen, Faaborg, Jost, Richardson, Maglo}

Set membership:

Set membership is indicated with the symbol "$\in$" (a stylized epsilon).

Gauker $\in$ {Gauker, Martin, Robinson, Polger, Skipper, Faaborg, Jost, Richardson, Maglo, Carbonell, Allen}.

Obama $\notin$ {Gauker, Martin, Robinson, Polger, Skipper, Faaborg, Jost, Richardson, Maglo, Carbonell, Allen}.

The empty set:

$\varnothing = \{\ \}$

Do not confuse this with the set *containing* the empty set: $\{\ \varnothing\ \}$

Unions: "$A \cup B$" stands for the *union* of sets A and B.

For example, $\{1, 5, 9\} \cup \{3, 9, 12\} = \{1, 3, 5, 9, 12\}$.

$\bigcup_{i=1}^{\infty} A_i$ is the union of the infinite series of sets $A_1, A_2, A_3, \ldots$

Subset: "$A \subseteq B$" means that A is a subset of B.

$\{1, 5\} \subseteq \{1, 2, 5, 7, 9\}$.
$\{1, 5\} \subseteq \{1, 5\}$.

n-tuples

A pair (or 2-tuple): $\langle$Gauker, Martin$\rangle$

A triple (or 3-tuple): $\langle$Gauker, Martin, Robinson$\rangle$

A one-tuple: $\langle$Gauker$\rangle$

Order matters: $\langle$Gauker, Martin$\rangle \neq \langle$Martin, Gauker$\rangle$.

Functions

A function is a thing with inputs and outputs, and for each input there is exactly one output.

Square(x) = y. Square(3) = 9. $x^2 = y$.

+(x, y) = z. +(2, 5) = 7. This can be abbreviated: x + y = z. The input is a pair $\langle x, y \rangle$.

fatherof(x) = y. fatherof(Beau) = Lloyd.

The *domain* of a function is the set of things that are inputs to the function.

The *range* of a function is the set of things that are outputs of the function for some input to the function.

A function can be thought of as a set of pairs:
For example, the addition function over positive integers =
$\{\langle\langle 1, 1\rangle, 2\rangle, \langle\langle 1, 2\rangle, 3\rangle, \langle\langle 2, 1\rangle, 3\rangle, \langle\langle 2, 2\rangle, 4\rangle, \ldots.\}$

The identity relation on **D**

The identity relation on a set **D** is the smallest set of ordered pairs such that for every object $o \in \mathbf{D}$, $\langle o, o \rangle$ is a member.

$\{\langle o_1, o_1\rangle, \langle o_2, o_2\rangle, \langle o_3, o_3\rangle, \ldots.\}$

Definition of a structure
(Other terms for structures: "model", "interpretation".)

Each structure is a *pair*:

$\mathbf{M} = \langle \mathbf{D_M}, \Sigma_\mathbf{M} \rangle$, or just $\langle \mathbf{D}, \Sigma \rangle$, for short.

$\mathbf{D_M}$, the *domain* of **M**, is a set of objects, e.g., $\{o_7, o_2, o_{66}, \ldots\}$.

The domain must be *nonempty*.

$\Sigma_{\mathbf{M}}$ is an *interpretation*:

For each name **n** of **L**, $\Sigma_{\mathbf{M}}(\mathbf{n}) \in \mathbf{D_M}$.
(That is, $\Sigma_{\mathbf{M}}$ assigns to each name a member of $\mathbf{D_M}$.)

Example: $\Sigma_{\mathbf{M}}(\text{“b”}) = o_5$, assuming $o_5 \in \mathbf{D_M}$.

For each *n*-place predicate **P**, $\Sigma_{\mathbf{M}}(\mathbf{P})$ = a set of *n*-tuples whose members are all members of $\mathbf{D_M}$.

For example, $\Sigma_{\mathbf{M}}(\text{“Larger”}) = \{\langle o_2, o_5\rangle, \langle o_5, o_6\rangle, \langle o_2, o_6\rangle\}$, assuming that o_2, o_5, o_6 are all members of $\mathbf{D_M}$.

For example, $\Sigma_{\mathbf{M}}(\text{“Cube”}) = \{\langle o_2\rangle, \langle o_7\rangle\}$.

$\Sigma_{\mathbf{M}}(\text{“}\mathbf{=}\text{”})$ = the identity relation on $\mathbf{D_M}$, as defined above.

Variable assignments g in **M**.

The domain of a variable assignment *g* in **M** is some subset of the set of variables of the language. The domain may be the empty set, ∅.

For each variable **v** in the domain of *g*,
$g(\mathbf{v}) \in \mathbf{D_M}$.

Examples: $g(\text{“x”}) = o_3$, $g(\text{“z”}) = o_7$, assuming o_3, $o_7 \in \mathbf{D_M}$.

Strictly speaking, to mark the dependence of *g* on $\mathbf{D_M}$, we should write $g_{\mathbf{M}}$, but for typographical simplicity, I omit the subscript (and I will soon start omitting the subscripts elsewhere as well).

Variants of variable assignments

$g[\mathbf{v}/o]$ is the variable assignment just like *g* except that $g[\mathbf{v}/o]$ assigns o to **v** instead of whatever *g* assigns to **v**.

Let’s say that $g[\mathbf{v}/o]$ is the “**v**-o variant of *g*”.

Example:

Suppose the domain of g is {“x”, “y”, “z”}, and

g(“x”) = o_1.
g(“y”) = o_2.
g(“z”) = o_3.

In that case, g[“y”/o_4] (the “y”-o_4 variant of g) is the following function:

g[“y”/o_4](“x”) = o_1.
g[“y”/o_4](“y”) = o_4.
g[“y”/o_4](“z”) = o_3.

And g[“y”/o_4][“z”/o_2] is the following function:

g[“y”/o_4][“z”/o_2](“x”) = o_1.
g[“y”/o_4][“z”/o_2](“y”) = o_4.
g[“y”/o_4][“z”/o_2](“z”) = o_2.

And g[“y”/o_4][“y”/o_5] is the following function:

g[“y”/o_4][“y”/o_5](“x”) = o_1.
g[“y”/o_4][“y”/o_5](“y”) = o_5.
g[“y”/o_4][“y”/o_5](“z”) = o_3.

In this last example, ‘g[“y”/o_4][“y”/o_5]’ is the name of the function. What we write after the name of the function between round parentheses is the input to the function.

Term assignments

Names and variables are both called "terms".

Where **t** is a term and *g* is some variable assignment in **M**,

$$h(\mathbf{t}) = \begin{cases} \Sigma_{\mathbf{M}}(\mathbf{t}) \text{ if } \mathbf{t} \text{ is a name.} \\ g(\mathbf{t}) \text{ if } \mathbf{t} \text{ is a variable.} \end{cases}$$

h is a "term assignment for **M** and *g*".

So, in other words a term assignment *h combines* the functions $\Sigma_{\mathbf{M}}$ and *g*. Strictly speaking, to mark the dependence of *h* on **M**, we should write $h_{\mathbf{M}}$, but to avoid notational clutter we will not.

For example:

Suppose:

$\Sigma_{\mathbf{M}}(\text{“b”}) = o_2$.
$g(\text{“z”}) = o_7$.

In that case,

$h(\text{“b”}) = o_2$.
$h(\text{“z”}) = o_7$.

And,

$\langle h(\text{“b”}), h(\text{“z”})\rangle = \langle \Sigma_{\mathbf{M}}(\text{“b”}), g(\text{“z”})\rangle = \langle o_2, o_7\rangle$.

We will also have variants of such term assignments. Thus:

$$h[\mathbf{v}/o](\mathbf{t}) = \begin{cases} \Sigma_{\mathbf{M}}(\mathbf{t}) \text{ if } \mathbf{t} \text{ is a name.} \\ g[\mathbf{v}/o](\mathbf{t}) \text{ if } \mathbf{t} \text{ is a variable.} \end{cases}$$

Barwise and Etchemendy notation

For users of Barwise and Etchemendy, I should note that the present notation differs from theirs in several ways.

Instead of $\mathbf{M}$, they write $\mathfrak{M}$.

Instead of $\mathbf{D_M}$, they write $\mathfrak{M}(\forall)$ and $D^{\mathfrak{M}}$.

Instead of $\Sigma_{\mathbf{M}}(\text{“Cube”})$, they write $\text{“Cube”}^{\mathfrak{M}}$ (but they omit the quotation marks).

And instead of $\Sigma_{\mathbf{M}}(\mathbf{P})$, they write $\mathbf{P}^{\mathfrak{M}}$.

Instead of $h(\mathbf{t})$, they write: $\|\mathbf{t}\|^{\mathfrak{M}}_{g}$

The good thing about their notation is that it makes explicit the relativity of term assignments to *g*.

From now on, I will drop the subscript “**M**” on “$\mathbf{D_M}$” and “$\Sigma_{\mathbf{M}}$”, just to reduce the clutter. (But don’t forget that it’s “really there”.)

Satisfaction of an atomic formula by a variable assignment in a structure:

Towards defining the conditions under which a variable assignment satisfies a formula in a structure, we first define the conditions under which a variable assignment satisfies an *atomic* formula in a structure:

g satisfies $\mathbf{R}(\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_n})$ in $\mathbf{M}$ if and only if:

$\langle h(\mathbf{t_1}), h(\mathbf{t_2}), \ldots, h(\mathbf{t_n})\rangle \in \Sigma_{\mathbf{M}}(\mathbf{R})$.

Example: "Larger(b, y)"

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where

$\mathbf{D} = \{o_1, o_2, o_3\}$.

Suppose $\Sigma(\text{“b”}) = o_3$, and
$\Sigma(\text{“Larger”}) = \{\langle o_2, o_3\rangle, \langle o_3, o_1\rangle, \langle o_2, o_1\rangle\}$, and
$g(\text{“y”}) = o_1$.

In that case, $\langle h(\text{“b”}), h(\text{“y”})\rangle = \langle \Sigma(\text{“b”}), g(\text{“y”})\rangle = \langle o_3, o_1\rangle \in \Sigma(\text{“Larger”})$.

So *g* satisfies "Larger(b, y)" in **M**.

Another example: "Adjoins(x, y)"

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where

$\mathbf{D} = \{o_1, o_2, o_3\}$.

$\Sigma(\text{“Adjoins”}) = \{\langle o_2, o_3\rangle, \langle o_3, o_2\rangle\}$.

$g(\text{“x”}) = o_1$.
$g(\text{“y”}) = o_2$.

Thus, $\langle h(\text{“x”}), h(\text{“y”})\rangle = \langle g(\text{“x”}), g(\text{“y”})\rangle = \langle o_1, o_2\rangle \notin \Sigma(\text{“Adjoins”})$.

So *g* does not satisfy "Adjoins(x, y)" in **M**.

Satisfaction of a disjunction by a variable assignment in a structure

To illustrate the manner in which we can define the conditions under which variable assignments satisfy complex formulas in a structure, consider the case of disjunctions:

g satisfies **(Q ∨ R)** in **M** if and only if either *g* satisfies **Q** in **M** or *g* satisfies **R** in **M**.

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where

$\mathbf{D} = \{o_1, o_2, o_3\}$.

$\Sigma(\text{“Cube”}) = \{\langle o_2 \rangle, \langle o_3 \rangle\}$.
$\Sigma(\text{“Tet”}) = \{\langle o_1 \rangle\}$.
$\Sigma(\text{“a”}) = o_2$.

$g(\text{“x”}) = o_2$.

$\langle h(\text{“a”}) \rangle = \langle \Sigma(\text{“a”}) \rangle = \langle o_2 \rangle \in \Sigma(\text{“Cube”})$. So g satisfies “Cube(a)” in **M**.

$\langle h(\text{“x”}) \rangle = \langle g(\text{“x”}) \rangle = \langle o_2 \rangle \notin \Sigma(\text{“Tet”})$. So g does not satisfy “Tet(x)” in **M**.

So either g satisfies “Cube(a)” in **M** or g satisfies “Tet(x)” in **M**.

So g satisfies “(Cube(a) ∨ Tet(x))” in **M**.

Satisfaction of an existential quantification by variable assignment in a structure

To illustrate the manner in which we can define the conditions under which variable assignments satisfy quantified formulas in a structure, consider the case of existential quantifications:

g satisfies $\exists \mathbf{v}\mathbf{Q}$ in **M** if and only if for some $o \in \mathbf{D}$, $g[\mathbf{v}/o]$ satisfies **Q** in **M**.

Example:

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where

$\mathbf{D} = \{o_1, o_2, o_3, o_4\}$.

$\Sigma(\text{“Cube”}) = \{\langle o_2 \rangle, \langle o_4 \rangle\}$.

$g(\text{“x”}) = o_1$.
$g[\text{“x”}/o_2](\text{“x”}) = o_2$.

$\langle h[\text{“x”}/o_2](\text{“x”}) \rangle = \langle g[\text{“x”}/o_2](\text{“x”}) \rangle = \langle o_2 \rangle \in \Sigma(\text{“Cube”})$.

So g[“x”/o_2] satisfies “Cube(x)” in **M**.
And $o_2 \in$ **D**.

So for some o $\in$ **D**, g[“x”/o] satisfies “Cube(x)” in **M**.

So g satisfies “∃xCube(x)” in **M**.

Another example:

Suppose **M** = ⟨**D**, Σ⟩, where

D = {o_1, o_2, o_3, o_4}.

Σ(“Adjoins”) = {⟨o_1, o_2⟩, ⟨o_2, o_1⟩}.

Σ(“b”) = o_3.

g(“y”) = o_2

There is no object o $\in$ **D** such that

⟨h[“y”/o](“b”), h[“y”/o](“y”)⟩ $\in$ Σ(“Adjoins”).

For example,

⟨h[“y”/o_1](“b”), h[“y”/o_1](“y”)⟩ =

⟨Σ(“b”), g[“y”/o_1](“y”)⟩ =

⟨o_3, o_1⟩ $\notin$ Σ(“Adjoins”).

So, g does not satisfy “∃yAdjoins(b, y)” in **M**.

Recursive Definitions
(Also called "inductive definitions".)

We now want to put together a number of stipulations concerning satisfaction, such as those above, to form a complete definition of satisfaction. The definition of satisfaction will be a "recursive definition". So let's first get a sense of what those are like:

Some ordinary definitions:

> A number *x* is a *positive prime* if and only if *x* is greater than 1 and *x* is divisible only by 1 and itself.
>
> *x* is a *shoathanger* if and only if either (i) *x* is a sheep or (ii) *x* is a coat hanger.

A circular definition (bad! not really a "definition" at all):

> A thing *x* is a *schmuck* if and only if either (i) *x* is a liar or (ii) *x* is a friend of a *schmuck*.
>
> The problem with this "definition" is that if something is not a liar, and none of its frends are liars, and none of the friends of its friends are liars, and so on, then we can draw no conclusions about whether it is a schmuck or not.

A recursive definition:

S is a *string* if and only if either
(i) *S* = "o", or
(ii) *S* = "o" followed by a string.

> So, "o" is a string.
> "oo" is a string, since "o" is a string and "oo" = "o" followed by a string, namely, "o".
> "ooo" is a string, since "ooo" = "o" followed by a string, namely, "oo".
> And so on.
> But "ooxo" is not a string, because it is not "o" followed by a string; because "oxo" is not a string, because it is not "o" followed by a string; "xo" is not a string, because it is neither "o" nor "o" followed by a string.

Notice in this example that, although the term "string" occurs on the right-hand side, the definition is not circular, because whenever we want to know whether something is a string, we get back to the question of whether something is "o" or starts with "o", which is a question we can answer just by looking.

The Definition of Satisfaction (ta da!):

For every wff **P** and every structure **M** and every variable assignment *g* in **M**, *g satisfies* **P** in **M** if and only if either:

(i) $\mathbf{P} = \mathbf{R}(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n)$, where **R** is an *n*-ary predicate and $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n$ are *n* terms, and $\langle h(\mathbf{t}_1), h(\mathbf{t}_2), \ldots, h(\mathbf{t}_n)\rangle \in \Sigma_{\mathbf{M}}(\mathbf{R})$, or

(ii) $\mathbf{P} = \neg \mathbf{Q}$, where **Q** is a wff, and
g does *not* satisfies **Q** in **M**, or

(iii) $\mathbf{P} = (\mathbf{Q} \wedge \mathbf{R})$, where **Q** and **R** are wffs, and
both *g* satisfies **Q** in **M** and *g* satisfies **R** in **M**, or

(iv) $\mathbf{P} = (\mathbf{Q} \vee \mathbf{R})$, where **Q** and **R** are wffs, and
either *g* satisfies **Q** in **M** or *g* satisfies **R** in **M**, or

(v) $\mathbf{P} = (\mathbf{Q} \rightarrow \mathbf{R})$, where **Q** and **R** are wffs, and
either *g* does not satisfy **Q** in **M** or *g* satisfies **R** in **M**, or

(vi) $\mathbf{P} = (\mathbf{Q} \leftrightarrow \mathbf{R})$, where **Q** and **R** are wffs, and
either *g* satisfies both **Q** and **R** in **M** or *g* satisfies neither **Q** nor **R** in **M**, or

(vii) $\mathbf{P} = \forall \mathbf{v}\mathbf{Q}$, where **Q** is a wff, and
for every object $o \in \mathbf{D}_{\mathbf{M}}$, $g[\mathbf{v}/o]$ satisfies **Q** in **M**, or

(viii) $\mathbf{P} = \exists \mathbf{v}\mathbf{Q}$, where **Q** is a wff, and
for some object $o \in \mathbf{D}_{\mathbf{M}}$, $g[\mathbf{v}/o]$ satisfies **Q** in **M**.

An alternative formulation

Suppose **P** is a wff, **M** is a structure, and *g* is a variable assignment in **M**.

1. Suppose $\mathbf{P} = \mathbf{R}(\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_n})$, where **R** is an *n*-ary predicate and $\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_n}$ are *n* terms.
   Then *g* satisfies **P** in **M** if and only if $\langle h(\mathbf{t_1}), h(\mathbf{t_2}), \ldots, h(\mathbf{t_n})\rangle \in \Sigma_{\mathbf{M}}(\mathbf{R})$.

2. Suppose $\mathbf{P} = \neg \mathbf{Q}$, where **Q** is a wff.
   Then *g* satisfies **P** in **M** if and only if *g* does *not* satisfy **Q** in **M**.

3. Suppose $\mathbf{P} = (\mathbf{Q} \wedge \mathbf{R})$, where **Q** and **R** are wffs.
   Then *g* satisfies **P** in **M** if and only if
   both *g* satisfies **Q** in **M** and *g* satisfies **R** in **M**.

4. Suppose $\mathbf{P} = (\mathbf{Q} \vee \mathbf{R})$, where **Q** and **R** are wffs.
   Then *g* satisfies **P** in **M** if and only if
   either *g* satisfies **Q** in **M** or *g* satisfies **R** in **M**.

5. Suppose $\mathbf{P} = (\mathbf{Q} \rightarrow \mathbf{R})$, where **Q** and **R** are wffs.
   Then *g* satisfies **P** in **M** if and only if
   either *g* does not satisfy **Q** in **M** or *g* satisfies **R** in **M**.

6. Suppose $\mathbf{P} = (\mathbf{Q} \leftrightarrow \mathbf{R})$, where **Q** and **R** are wffs.
   Then *g* satisfies **P** in **M** if and only if either *g* satisfies both **Q** and **R** in **M**, or *g* satisfies neither **Q** nor **R** in **M**.

7. Suppose $\mathbf{P} = \forall \mathbf{v}\mathbf{Q}$, where **Q** is a wff.
   Then *g* satisfies **P** in **M** if and only if
   for every object $o \in \mathbf{D_M}$, $g[\mathbf{v}/o]$ satisfies **Q** in **M**.

8. Suppose $\mathbf{P} = \exists \mathbf{v}\mathbf{Q}$, where **Q** is a wff.
   Then *g* satisfies **P** in **M** if and only if
   for some object $o \in \mathbf{D_M}$, $g[\mathbf{v}/o]$ satisfies **Q** in **M**.

We can *call* this the "definition of satisfaction" too, although actually it is a list of "axioms".

The *empty variable assignment*, $g_\varnothing$, is the variable assignment whose domain is $\varnothing$.

The definition of truth in a structure:

A sentence **S** is *true* in a structure **M** if and only if
the empty variable assignment $g_{\varnothing}$ satisfies **S** in **M**.

Students often find this definition troubling. How can anything of importance depend on the *empty* variable assignment? Well, the empty variable assignment gives us a "hook" on which to spin variations as we consider the subformulas of the sentence in question.

Example:

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where
$\mathbf{D} = \{o_1, o_2\}$,
$\Sigma(\text{“Cube”}) = \{\langle o_1 \rangle\}$,
$\Sigma(\text{“Tet”}) = \varnothing$.

So $\langle o_1 \rangle \in \Sigma(\text{“Cube”})$.
So $\langle g_{\varnothing}[\text{“x”}/o_1](\text{“x”}) \rangle \in \Sigma(\text{“Cube”})$.
So $\langle h_{\varnothing}[\text{“x”}/o_1](\text{“x”}) \rangle \in \Sigma(\text{“Cube”})$.
By clause 1 in the def. of satisfaction, $g_{\varnothing}[\text{“x”}/o_1]$ satisfies “Cube(x)” in **M**.
So there exists an $o \in \mathbf{D}$ such that $g_{\varnothing}[\text{“x”}/o]$ satisfies “Cube(x)” in **M**.
By clause 8 in the def. of satisfaction, $g_{\varnothing}$ satisfies “∃xCube(x)” in **M**.
So “∃xCube(x)” is true in **M**.

$\langle o_1 \rangle \notin \Sigma(\text{“Tet”})$.
$\langle g_{\varnothing}[\text{“x”}/o_1](\text{“x”}) \rangle \notin \Sigma(\text{“Tet”})$.
$\langle h_{\varnothing}[\text{“x”}/o_1](\text{“x”}) \rangle \notin \Sigma(\text{“Tet”})$.
By clause 1, $g_{\varnothing}[\text{“x”}/o_1]$ does not satisfy “Tet(x)” in **M**.
$\langle o_2 \rangle \notin \Sigma(\text{“Tet”})$.
$\langle g_{\varnothing}[\text{“x”}/o_2](\text{“x”}) \rangle \notin \Sigma(\text{“Tet”})$.
$\langle h_{\varnothing}[\text{“x”}/o_2](\text{“x”}) \rangle \notin \Sigma(\text{“Tet”})$.
By clause 1, $g_{\varnothing}[\text{“x”}/o_2]$ does not satisfy “Tet(x)” in **M**.
So there is no $o \in \mathbf{D}$ such that $g_{\varnothing}[\text{“x”}/o]$ satisfies “Tet(x)” in **M**.
By clause 8, $g_{\varnothing}$ does not satisfy “∃xTet(x)” in **M**.
So “∃xTet(x)” is not true in **M**.

More examples:

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where
$\mathbf{D} = \{o_1, o_2\}$,
$\Sigma(\text{“Cube”}) = \{\langle o_1 \rangle\}$,
$\Sigma(\text{“a”}) = o_1$. $\Sigma(\text{“b”}) = o_2$.

So $\langle o_1 \rangle \in \Sigma(\text{“Cube”})$.
So $\langle \Sigma(\text{“a”}) \rangle \in \Sigma(\text{“Cube”})$.
So $\langle h_\varnothing(\text{“a”}) \rangle \in \Sigma(\text{“Cube”})$.
By clause 1, $g_\varnothing$ satisfies “Cube(a)” in **M**.
So by the definition of truth, “Cube(a)” is true in **M**.

So $\langle o_2 \rangle \notin \Sigma(\text{“Cube”})$.
So $\langle \Sigma(\text{“b”}) \rangle \notin \Sigma(\text{“Cube”})$.
So $\langle h_\varnothing(\text{“b”}) \rangle \notin \Sigma(\text{“Cube”})$.
By clause 1, $g_\varnothing$ does not satisfy “Cube(b)” in **M**.
So by the definition of truth, “Cube(b)” is not true in **M**.

By clause 2, $g_\varnothing$ satisfies “¬Cube(b)” in **M**.
So by the definition of truth, “¬Cube(b)” is true in **M**.

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where
$\mathbf{D} = \{o_1, o_2\}$,
$\Sigma(\text{“Sameshape”}) = \{\langle o_1, o_1 \rangle, \langle o_2, o_2 \rangle, \langle o_1, o_2 \rangle, \langle o_2, o_1 \rangle\}$
$\Sigma(\text{“a”}) = o_1$.

$\langle o_1, o_1 \rangle \in \Sigma(\text{“Sameshape”})$.
$\langle g_\varnothing[\text{“x”}/o_1](\text{“x”}), \Sigma(\text{“a”}) \rangle \in \Sigma(\text{“Sameshape”})$.
$\langle h_\varnothing[\text{“x”}/o_1](\text{“x”}), h_\varnothing[\text{“x”}/o_1](\text{“a”}) \rangle \in \Sigma(\text{“Sameshape”})$.
By clause 1, $g_\varnothing[\text{“x”}/o_1]$ satisfies “Sameshape(x, a)” in **M**.

$\langle o_2, o_1 \rangle \in \Sigma(\text{“Sameshape”})$.
$\langle g_\varnothing[\text{“x”}/o_2](\text{“x”}), \Sigma(\text{“a”}) \rangle \in \Sigma(\text{“Sameshape”})$.
$\langle h_\varnothing[\text{“x”}/o_2](\text{“x”}), h_\varnothing[\text{“x”}/o_2](\text{“a”}) \rangle \in \Sigma(\text{“Sameshape”})$.
By clause 1, $g_\varnothing[\text{“x”}/o_2]$ satisfies “Sameshape(x, a)” in **M**.

So for all $o \in \mathbf{D}$, $g_\varnothing[\text{“x”}/o]$ satisfies “Sameshape(x, a)” in **M**.
So by clause 7, $g_\varnothing$ satisfies “∀xSameshape(x, a)” in **M**.
So by the definition of truth, “∀xSameshape(x, a)” is true in **M**.

The definition of first-order validity:

An argument is *first-order valid* (i.e., the conclusion is a *first-order consequence* of the premises) if and only if for each *structure* **M** of the language, if the premises are all true in **M**, then the conclusion is true in **M** too.

Examples:

Example 1: The inference from "∃xCube(x)" to "Cube(b)" is *not* first-order valid.

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where
$\mathbf{D} = \{o_1, o_2\}$,
$\Sigma(\text{“Cube”}) = \{\langle o_1 \rangle\}$,
$\Sigma(\text{“b”}) = o_2$.

We have already seen that "∃xCube(x)" is true in **M**, and we have already seen that "Cube(b)" is not true in **M**.

Example 2: The inference from "Cube(b)" to "∃xCube(x)" *is* first-order valid:

Suppose, for arbitrary structure **M**, "Cube(b)" is true in **M**.
By the definition of truth, $g_\varnothing$ satisfies "Cube(b)" in **M**.
By clause 1 in the definition of satisfaction, $\langle h_\varnothing(\text{“b”}) \rangle \in \Sigma(\text{“Cube”})$.
So there is an object $o \in \mathbf{D}$ such that $\langle o \rangle \in \Sigma(\text{“Cube”})$.
So there is an $o \in \mathbf{D}$ such that $\langle h_\varnothing[\text{“x”}/o](\text{“x”}) \rangle \in \Sigma(\text{“Cube”})$.
So by clause 1, there is an $o \in \mathbf{D}$ such that $g_\varnothing[\text{“x”}/o]$ satisfies "Cube(x)" in **M**.
So by clause 8, $g_\varnothing$ satisfies "∃xCube(x)" in **M**.
So "∃xCube(x)" is true in **M**.
But **M** was arbitrary.
So for all structures **M**, if "Cube(b)" is true in **M**, then "∃xCube(x)" is true in **M** as well.

Example 3: Prove that the following argument *is* first-order valid:

∀x(F(x) → G(x))
∃xF(x)

∃xG(x)

Suppose "$\forall x(F(x) \rightarrow G(x))$" and "$\exists x F(x)$" are true in arbitrary **M**.
$g_{\varnothing}$ satisfies "$\forall x(F(x) \rightarrow G(x))$" and "$\exists x F(x)$" in **M**.
For some o ∈ **D**, $g_{\varnothing}$["x"/o] satisfies "F(x)" in **M**.
Suppose a ∈ **D** and $g_{\varnothing}$["x"/a] satisfies "F(x)" in **M**.
For all o ∈ **D**, $g_{\varnothing}$["x"/o] satisfies "(F(x) → G(x))" in **M**.
So, $g_{\varnothing}$["x"/a] satisfies "(F(x) → G(x))" in **M**.
Either $g_{\varnothing}$["x"/a] does not satisfy "F(x)" or $g_{\varnothing}$["x"/a] satisfies "G(x)" in **M**.
So, $g_{\varnothing}$["x"/a] satisfies "G(x)" in **M**.
So, for some o ∈ **D**, $g_{\varnothing}$["x"/o] satisfies "$\exists x G(x)$" in **M**.
$g_{\varnothing}$ satisfes "$\exists x G(x)$" in **M**.
"$\exists x G(x)$" is true in **M**.

Example 4: Prove that the following argument is *not* first-order valid:

$(\exists x F(x) \wedge \exists x G(x))$

$\exists x(F(x) \wedge G(x))$

Suppose **M** = ⟨**D**, Σ⟩, where
D = {a, b}, and
Σ("F") = {⟨a⟩}, and
Σ("G") = {⟨b⟩}.

⟨ $g_{\varnothing}$["x"/a]("x")⟩ ∈ Σ("F").
⟨$h_{\varnothing}$["x"/a]("x")⟩ ∈ Σ("F").
$g_{\varnothing}$["x"/a] satisfies "F(x)" in **M**.
For some o ∈ **D**, $g_{\varnothing}$["x"/o] satisfies "F(x)" in **M**.
$g_{\varnothing}$ satisfies $\exists x F(x)$ in **M**.

⟨ $g_{\varnothing}$["x"/b]("x")⟩ ∈ Σ("G").
⟨ $h_{\varnothing}$["x"/b]("x")⟩ ∈ Σ("G").
$g_{\varnothing}$["x"/b] satisfies "G(x)" in **M**.
For some o ∈ **D**, $g_{\varnothing}$["x"/o] satisfies "G(x)" in **M**.
$g_{\varnothing}$ satisfies "$\exists x G(x)$" in **M**.

$g_{\varnothing}$ satisfies "$\exists x F(x)$" in **M** and $g_{\varnothing}$ satisfies "$\exists x G(x)$" in **M**.
$g_{\varnothing}$ satisfies "$(\exists x F(x) \wedge \exists x G(x))$" in **M**.
"$(\exists x F(x) \wedge \exists x G(x))$" is true in **M**.

$\langle g_{\varnothing}[\text{“x”/b}](\text{“x”})\rangle \notin \Sigma(\text{“F”})$.
$\langle h_{\varnothing}[\text{“x”/b}](\text{“x”})\rangle \notin \Sigma(\text{“F”})$.
$g_{\varnothing}[\text{“x”/b}]$ does not satisfy “F(x)” in **M**.
$g_{\varnothing}[\text{“x”/b}]$ does not satisfy “(F(x) ∧ G(x))” in **M**.

$\langle g_{\varnothing}[\text{“x”/a}](\text{“x”})\rangle \notin \Sigma(\text{“G”})$.
$\langle h_{\varnothing}[\text{“x”/a}](\text{“x”})\rangle \notin \Sigma(\text{“G”})$.
$g_{\varnothing}[\text{“x”/a}]$ does not satisfy “G(x)” in **M**.
$g_{\varnothing}[\text{“x”/a}]$ does not satisfy “(F(x) ∧ G(x))” in **M**.

For all o ∈ **D**, $g_{\varnothing}[\text{“x”/o}]$ does not satisfy “(F(x) ∧ G(x))” in **M**.
$g_{\varnothing}$ does not satisfy “∃x(F(x) ∧ G(x))” in **M**.
“∃x(F(x) ∧ G(x))” is not true in **M**.

NOTE: In all subsequent examples, I will omit all quotation marks. However, do not forget that “really” they are there.

Example 5: Show that the following argument *is* first-order valid:

∃x∀yR(x, y)

∀y∃xR(x, y)

Suppose ∃x∀yR(x, y) is true for arbitrary **M**.
$g_{\varnothing}$ satisfies ∃x∀yR(x, y) in **M**.
For some o ∈ **D**, $g_{\varnothing}$[x/o] satisfies ∀yR(x, y) in **M**.
Suppose a ∈ **D** and $g_{\varnothing}$[x/a] satisfies ∀yR(x, y) in **M**.

For all o ∈ **D**, $g_{\varnothing}$[x/a][y/o] satisfies R(x, y) in **M**.
So, for all o ∈ **D**, $g_{\varnothing}$[y/o][x/a] satisfies R(x, y) in **M**. (Notice the difference.)

For all o ∈ **D**, there is some o′ ∈ **D** such that $g_{\varnothing}$[y/o][x/o′] satisfies R(x, y) in **M**.
For all o ∈ **D**, $g_{\varnothing}$[y/o] satisfies ∃xR(x, y) in **M**.
$g_{\varnothing}$ satisfies ∀y∃xR(x, y) in **M**.

∀y∃xR(x, y) is true in **M**.

Example 6: Show that the following argument is *not* first-order valid:

$\forall x \exists y R(x, y)$

$\exists y \forall x R(x, y)$

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where
$\mathbf{D} = \{a, b\}$, and
$\Sigma(R) = \{\langle a, a\rangle, \langle b, b\rangle\}$.

$\langle g_{\varnothing}[x/a][y/a](x), g_{\varnothing}[x/a][y/a](y)\rangle \in \Sigma(R)$.
$g_{\varnothing}[x/a][y/a]$ satisfies $R(x, y)$ in $\mathbf{M}$.
$g_{\varnothing}[x/a]$ satisfies $\exists y R(x, y)$ in $\mathbf{M}$.

$\langle g_{\varnothing}[x/b][y/b](x), g_{\varnothing}[x/b][y/b](y)\rangle \in \Sigma(R)$.
$g_{\varnothing}[x/b][y/b]$ satisfies $R(x, y)$ in $\mathbf{M}$.
$g_{\varnothing}[x/b]$ satisfies $\exists y R(x, y)$ in $\mathbf{M}$.

For all $o \in \mathbf{D}$, $g_{\varnothing}[x/o]$ satisfies $\exists y R(x, y)$ in $\mathbf{M}$.
$g_{\varnothing}$ satisfies $\forall x \exists y R(x, y)$ in $\mathbf{M}$.
$\forall x \exists y R(x, y)$ is true in $\mathbf{M}$.

$\langle g_{\varnothing}[y/a][x/b](x), g_{\varnothing}[y/a][x/b](y)\rangle \notin \Sigma(R)$.
$g_{\varnothing}[y/a][x/b]$ does not satisfy $R(x, y)$ in $\mathbf{M}$.
It is not the case that for all $o \in \mathbf{D}$, $g_{\varnothing}[y/a][x/o]$ satisfies $R(x, y)$ in $\mathbf{M}$.
$g_{\varnothing}[y/a]$ does not satisfy $\forall x R(x, y)$ in $\mathbf{M}$.

$\langle g_{\varnothing}[y/b][x/a](x), g_{\varnothing}[y/b][x/a](y)\rangle \notin \Sigma(R)$.
$g_{\varnothing}[y/b][x/a]$ does not satisfy $R(x, y)$ in $\mathbf{M}$.
It is not the case that for all $o \in \mathbf{D}$, $g_{\varnothing}[y/b][x/o]$ satisfies $R(x, y)$ in $\mathbf{M}$.
$g_{\varnothing}[y/b]$ does not satisfy $\forall x R(x, y)$ in $\mathbf{M}$.

For all $o \in \mathbf{D}$, $g_{\varnothing}[y/o]$ does not satisfy $\forall x R(x, y)$ in $\mathbf{M}$.
$g_{\varnothing}$ does not satisfy $\exists y \forall x R(x, y)$ in $\mathbf{M}$.
$\exists y \forall x R(x, y)$ is not true in $\mathbf{M}$.

Example 7: Show that the following argument *is* first-order valid:

$\neg\exists x(F(x) \wedge G(x))$

$\forall x(F(x) \rightarrow \neg G(x))$

Suppose, for arbitrary **M**, that $\neg\exists x(F(x) \wedge G(x))$ is true in **M**.
$g_{\varnothing}$ satisfies $\neg\exists x(F(x) \wedge G(x))$ in **M**.
$g_{\varnothing}$ does not satisfy $\exists x(F(x) \wedge G(x))$ in **M**.
There is no object $o \in \mathbf{D}$ such that $g_{\varnothing}[x/o]$ satisfies $(F(x) \wedge G(x))$ in **M**.
There is no object $o \in \mathbf{D}$ such that both $g_{\varnothing}[x/o]$ satisfies $F(x)$ in **M** and $g_{\varnothing}[x/o]$ satisfies $G(x)$ in **M**.
For all objects $o \in \mathbf{D}$, either $g_{\varnothing}[x/o]$ does not satisfy $F(x)$ in **M** or $g_{\varnothing}[x/o]$ does not satisfy $G(x)$ in **M**.
For all objects $o \in \mathbf{D}$, either $g_{\varnothing}[x/o]$ does not satisfy $F(x)$ in **M** or $g_{\varnothing}[x/o]$ satisfies $\neg G(x)$ in **M**.
For all objects $o \in \mathbf{D}$, $g_{\varnothing}[x/o]$ satisfies $(F(x) \rightarrow \neg G(x))$ in **M**.
$g_{\varnothing}$ satisfies $\forall x(F(x) \rightarrow \neg G(x))$ in **M**.
$\forall x(F(x) \rightarrow \neg G(x))$ is true in **M**.

Example 8: Show that the following argument is *not* first-order valid.

$(\forall xF(x) \rightarrow G(c))$

$\forall x(F(x) \rightarrow G(c))$

Suppose $\mathbf{M} = \langle \mathbf{D}, \Sigma \rangle$, where
$\mathbf{D} = \{a, b\}$, and
$\Sigma(F) = \{\langle a \rangle\}$, and
$\Sigma(G) = \{\langle b \rangle\}$.
$\Sigma(c) = a$.

$\langle g_{\varnothing}[x/b](x) \rangle \notin \Sigma(F)$.
$g_{\varnothing}[x/b]$ does not satisfy $F(x)$.
For some $o \in \mathbf{D}$, $g_{\varnothing}[x/o]$ does not satisfy $F(x)$.
$g_{\varnothing}$ does not satisfy $\forall xF(x)$ in **M**.
Either $g_{\varnothing}$ does not satisfy $\forall xF(x)$ in **M** or $g_{\varnothing}$ satisfies $G(c)$ in **M**.
$g_{\varnothing}$ satisfies $(\forall xF(x) \rightarrow G(c))$ in **M**.
$(\forall xF(x) \rightarrow G(c))$ is true in **M**.
$\langle g_{\varnothing}[x/a](x) \rangle \in \Sigma(F)$.

$g_{\varnothing}$[x/a] satisfies F(x) in **M**.
$\langle \Sigma(c) \rangle \notin \Sigma(G)$.
$\langle h_{\varnothing}[x/a](c) \rangle \notin \Sigma(G)$.
$g_{\varnothing}$[x/a] does not satisfy G(c) in **M**.
It is not the case that either $g_{\varnothing}$[x/a] does not satisfy F(x) in **M** or $g_{\varnothing}$[x/a] satisfies G(c) in **M**.
$g_{\varnothing}$[x/a] does not satisfy (F(x) → G(c)) in **M**.
It is not the case that for all o ∈ **D**, $g_{\varnothing}$[x/o] satisfies (F(x) → G(c)) in **M**.
$g_{\varnothing}$ does not satisfy ∀x(F(x) → G(c)) in **M**.
∀x(F(x) → G(c)) is not true in **M**.

Example 9: Show that the following argument is *not* first-order valid.

∀xLikes(x, d)

∀xLikes(x, x)

Suppose **M** = ⟨**D**, Σ⟩, where
D = {a, b}, and
Σ(Likes) = {⟨a, a⟩, ⟨b, a⟩}, and
Σ(d) = a.

$\langle g_{\varnothing}[x/a](x), \Sigma(d) \rangle \in \Sigma(Likes)$.
$\langle h_{\varnothing}[x/a](x), h_{\varnothing}[x/a](d) \rangle \in \Sigma(Likes)$.
So $g_{\varnothing}$[x/a] satisfies Likes(x, d).
$\langle g_{\varnothing}[x/b](x), \Sigma(d) \rangle \in \Sigma(Likes)$.
$\langle h_{\varnothing}[x/b](x), h_{\varnothing}[x/b](d) \rangle \in \Sigma(Likes)$.
So $g_{\varnothing}$[x/b] satisfies Likes(x, d)
For all o ∈ **D**, $g_{\varnothing}$[x/o] satisfies Likes(x, d).
$g_{\varnothing}$ satisfies ∀xLikes(x, d).
∀xLikes(x, d) is true in **M**.

$\langle g_{\varnothing}[x/b](x), g_{\varnothing}[x/b](x) \rangle \notin \Sigma(Likes)$.
$g_{\varnothing}$[x/b] does not satisfy Likes(x, x).
For some o ∈ **D**, $g_{\varnothing}$[x/o] does not satisfy Likes(x, x).
$g_{\varnothing}$ does not satisfy ∀xLikes(x, x).
∀xLikes(x, x) is not true in **M**.

Lesson 2: The Soundness Theorem for First-order Logic

Proof by induction

In proving general claims about sentences, arguments, etc., we will sometimes make use of the method of *proof by induction*, or *inductive proof*. So I want to begin by giving you that concept. I will start with a simple example, and then I will generalize in a vague way from that.

The example I will use has to do with three-valued logic. So first, before I give you the example of an inductive proof, I need to say a little about that.

Suppose we are dealing with a language $\mathcal{L}$ containing atomic sentences, the negation sign and the disjunction sign. Since we don't have quantifiers, we can ignore names and predicates and suppose that the atomic sentences are A, B, C, …. Sentences are built up from these atomic sentences in the usual way.

Say that atomic sentences have complexity 0.
If a sentence P has complexity *n*, then ¬P has complexity *n*+1.
If out of the two sentences P and Q, the complexity of the one with the greatest complexity is *n*, then the complexity of (P ∨ Q) is *n*+1.

For example, since A is an atomic sentence, ¬A has complexity 1.
The complexity of (B ∨ C) is 1.
The complexity of ((A ∨ B) ∨ ¬(B ∨ C)) is 3 (not 2, not 4!). Can you see why?

Say that a *three-valued assignment* σ for $\mathcal{L}$ is a function that takes atomic sentences of $\mathcal{L}$ as inputs and yields as outputs a member of the set {T, N, F}. "N" stands for *neither*. We think of sentences to which N is assigned as neither true nor false.

Let the truth tables for ¬ and ∨ be as follows:

P	¬P
T	F
N	N
F	T

P	Q	(P ∨ Q)
T	T	T
T	N	T
T	F	T
N	T	T
N	N	N
N	F	N
F	T	T
F	N	N
F	F	F

Or in other words:

∨	T	N	F
T	T	T	T
N	T	N	N
F	T	N	F

These tables simply display in graphic form the following definition of an evaluation V_σ.

For every three-valued assignment σ and every sentence S of $\mathcal{L}$,

(i) if S is atomic, then $V_\sigma(S) = \sigma(S)$, and

(ii) if $S = \neg P$, then
  (a) $V_\sigma(S) = T$ if and only if $V_\sigma(P) = F$, and
  (b) $V_\sigma(S) = F$ if and only if $V_\sigma(P) = T$, and
  (c) $V_\sigma(S) = N$ if and only if $V_\sigma(P) = N$, and

(iii) if $S = (P \vee Q)$, then
  (a) $V_\sigma(S) = T$ if and only if $V_\sigma(P) = T$ or $V_\sigma(Q) = T$, and
  (b) $V_\sigma(S) = F$ if and only if $V_\sigma(P) = F$ and $V_\sigma(Q) = F$, and
  (c) $V_\sigma(S) = N$ in every other case.

Now I state the following theorem

Theorem: Suppose σ and σ* are three-valued assignments for language $\mathcal{L}$.
And for all *atomic* sentences P of $\mathcal{L}$,
  if $\sigma(P) = T$, then $\sigma^*(P) = T$, and
  if $\sigma(P) = F$, then $\sigma^*(P) = F$.
  (That much is the "hypothesis" of the theorem.)
Then for *all* sentences P of $\mathcal{L}$ (what follows is "the thesis"),
  if $V_\sigma(P) = T$, then $V_{\sigma^*}(P) = T$, and
  if $V_\sigma(P) = F$, then $V_{\sigma^*}(P) = F$.

In other words, σ* does not change any of the assignments that σ makes, but σ* may assign T or F to *more* atomic sentences than σ does. Nonetheless, the theorem states that the switch from σ to σ* creates no new truth-value gaps among the compound sentences.

Proof: By induction "on the complexity of sentences".

Assume the hypothesis of the theorem (viz., that for all atomic sentences, …).

Basis: Show that the thesis holds for all atomic sentences (sentences of complexity 0). For each sentence P of complexity 0, $V_\sigma(\mathsf{P}) = \sigma(\mathsf{P})$ and $V_{\sigma*}(\mathsf{P}) = \sigma^*(\mathsf{P})$. So since σ^* assigns T or F if σ does, $V_{\sigma*}$ assigns T or F if V_σ does. In other words, the hypothesis, restricted to atomic sentences, implies the thesis, restricted to atomic sentences. (The basis clause for an inductive proof will not always be as easy as this!)

Induction hypothesis (IH): Suppose that the thesis holds for all sentences having a complexity less than or equal to *n*.

Induction step: Show that the thesis holds for all sentences having complexity equal to *n*+1.

(¬) Suppose $\mathsf{P} = \neg\mathsf{Q}$ has complexity *n*+1 and $V_\sigma(\neg\mathsf{Q}) = \mathsf{T}$. Then $V_\sigma(\mathsf{Q}) = \mathsf{F}$. But Q has complexity *n*; so by the induction hypothesis, $V_{\sigma*}(\mathsf{Q}) = \mathsf{F}$. So $V_{\sigma*}(\neg\mathsf{Q}) = \mathsf{T}$.

Suppose $\mathsf{P} = \neg\mathsf{Q}$ has complexity *n*+1 and $V_\sigma(\neg\mathsf{Q}) = \mathsf{F}$. Then $V_\sigma(\mathsf{Q}) = \mathsf{T}$. But Q has complexity *n*; so by the induction hypothesis, $V_{\sigma*}(\mathsf{Q}) = \mathsf{T}$. So $V_{\sigma*}(\neg\mathsf{Q}) = \mathsf{F}$.

(∨) Suppose $\mathsf{P} = (\mathsf{Q} \vee \mathsf{R})$ has complexity *n*+1 and $V_\sigma((\mathsf{Q} \vee \mathsf{R})) = \mathsf{T}$. Then either $V_\sigma(\mathsf{Q}) = \mathsf{T}$ or $V_\sigma(\mathsf{R}) = \mathsf{T}$. But either Q or R has complexity *n*, and the other has complexity less than or equal to *n*; so by the induction hypothesis, either $V_{\sigma*}(\mathsf{Q}) = \mathsf{T}$ or $V_{\sigma*}(\mathsf{R}) = \mathsf{T}$. So $V_{\sigma*}((\mathsf{Q} \vee \mathsf{R})) = \mathsf{T}$.

Suppose $\mathsf{P} = (\mathsf{Q} \vee \mathsf{R})$ has complexity *n*+1 and $V_\sigma((\mathsf{Q} \vee \mathsf{R})) = \mathsf{F}$. Then both $V_\sigma(\mathsf{Q}) = \mathsf{F}$ and $V_\sigma(\mathsf{R}) = \mathsf{F}$. But either Q or R has complexity *n*, and the other has complexity less than or equal to *n*; so by the induction hypothesis, both $V_{\sigma*}(\mathsf{Q}) = \mathsf{F}$ and $V_{\sigma*}(\mathsf{R}) = \mathsf{F}$. So $V_{\sigma*}((\mathsf{Q} \vee \mathsf{R})) = \mathsf{F}$.

End of Proof. (In other words, as this point, we consider the theorem proved.)

So here is the general pattern of a proof by induction:

When we are doing a proof by induction, we are always trying to say something about all of the members of a certain set of objects. (In the example, it might be the set of sentences in $\mathcal{L}$.) Moreover, the membership of that set is defined "inductively". That is, we begin the definition of the set by stipulating that certain *basic* objects belong to the

set. (In the example, that's the atomic sentences.) Then we say that if a certain number of things that we already know are in the set generate some other object, via one or another given member-generating functions, then that other object is in the set too. And that gives us the entire membership of the set. (In the example, the functions are "sentence-forming operations", which given one or more sentences, generate another sentence.)

If a set is defined inductively in this way, then we can prove that a thesis is true of every member of the set as follows: First, we show that the thesis holds for all basic members. That step is called the *basis*. (In the example, the hypothesis of the theorem was itself this part.) Then we suppose, for the sake of argument, that the thesis holds for some arbitrary members of the set, which for an arbitrary *m* we can also characterize as members that are generated by no more than *m* applications of the member-generating functions. That is the *induction hypothesis*. (In the example, we supposed that the thesis held for all sentences having complexity less than or equal to *n*. The maximum number of applications of sentence-forming operations that are necessary to form a sentence having complexity *n* is $(2n - 1)$ (try it!). So, in effect, we are stipulating a maximum number of sentence-forming operations.) Finally, we examine each of the member-generating functions and show that if we apply that function to the members of the set of which we are supposing the thesis holds, then the thesis holds as well for the members of the set that that function yields. That's the *induction step*. (In the example, we supposed that the thesis held for sentences having a maximum complexity of *n* and show that it holds for sentences having a maximum complexity of $n+1$, i.e., sentences that result from one more application of a sentence-forming operation.) On this basis, we conclude that the thesis holds for *every* member of the set.

Question: What gives us the right to assume that an inductive proof of this kind proves the theorem? Is that a logical truth? No, it's a fact about inductively defined sets (a fact of set theory).

Simplification of the language

In proving soundness and completeness, we will have to say something about each connective and quantifier in the language. So if we have fewer connectives and quantifiers, we will not have to say as much. So now I will stipulate that the language of first-order logic contains only the following logical symbols: $\neg$, $\rightarrow$, $\forall$, =. So we're dropping $\wedge$, $\vee$, $\leftrightarrow$ and $\exists$.

To be precise, we now define the language $\mathcal{L}$ as follows:

The vocabulary of $\mathcal{L}$:

Denumerably many individual constants: a, b, c, …
Denumerably, many individual variables: x, y, z, …
(Recall from Lesson 1 that individual constants and individual variables are called *terms*.)
For each *n*, denumerably many *n*-ary predicates: F, G, H, …
The identity symbol: =
The following sentential connectives: ¬, →
The universal quantifier: ∀
("Denumerably many" means: One for each of the natural numbers, 0, 1, 2, 3, ...)

The definitions of wff, bound variable and sentence of $\mathcal{L}$:

A string S of symbols from the vocabulary of $\mathcal{L}$ is a well-formed formula (wff) of $\mathcal{L}$ if and only if:
(a) $S = Pt_1t_2\ldots t_n$ and P is an *n*-ary predicate of $\mathcal{L}$ and $t_1, t_2, \ldots, t_n$ are terms of $\mathcal{L}$, or
(b) $S = \neg P$ and P is a well-formed formula of $\mathcal{L}$, or
(c) $S = (P \rightarrow Q)$ and P and Q are well-formed formulas of $\mathcal{L}$, or
(d) $S = \forall vP$ and P is a well-formed formula of $\mathcal{L}$ and v is a variable of $\mathcal{L}$ (v is said to be *bound* in this case).

Note: In this language we will not write parentheses after predicates. So in this language, an atomic sentence will be, for example, Hab, not H(a, b). There is a reason for this other than simplicity. Now when I do write something like P(v) that will stand for a formula that has a free variable v in it somewhere. If I write something like Pa/v, that will stand for the result of putting a in place of each free occurrence of v in P. For example, $(Hax \rightarrow \forall xGx)b/x$ is $(Hab \rightarrow \forall xGx)$.

A *sentence* of $\mathcal{L}$ is a well-formed formula of $\mathcal{L}$ in which every variable is bound.

Note: The elimination of ∧, ∨, ↔ and ∃ does not really reduce our "expressive power", because for every sentence in the old language containing one of these symbols there is a first-order equivalent sentence in that language that does not contain those symbols. That claim can be proved, by induction, but here I'll just state the general principles:

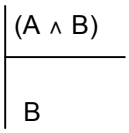
Any sentence of the form ∃xP is FO-equivalent to $\neg\forall x\neg P$.
Any sentence of the form $(P \vee Q)$ is FO-equivalent to $(\neg P \rightarrow Q)$.
Any sentence of the form $(P \wedge Q)$ is FO-equivalent to $\neg(P \rightarrow \neg Q)$.

Exercise: Consider the language containing ∧ and ∃ in addition to the vocabulary of $\mathcal{L}$. Show by induction on the complexity of sentences that for every sentence of that language there is an equivalent sentence that contains only the vocabulary of $\mathcal{L}$.

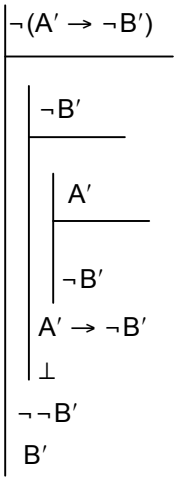
We can go on using ∧, ∨ and ∃, but we will think of sentences containing these symbols as merely abbreviations for sentences that do not contain them. We will also go on using ⊥, but now we will think of this as some particular contradiction that we can write using just ¬ and →, such as ¬(Fa → Fa).

Accordingly, we will not need the introduction and elimination rules for the connectives and quantifiers we have thrown away. If we throw away those introduction and elimination rules, will that mean that we cannot prove as many arguments? Yes and no. Yes, we cannot give proofs for arguments containing sentences containing symbols that are not in our language. But no, for any argument in the old language, if we could have proved it with the full set of Fitch rules, then we can prove its *translation* into the impoverished language using the impoverished set of rules.

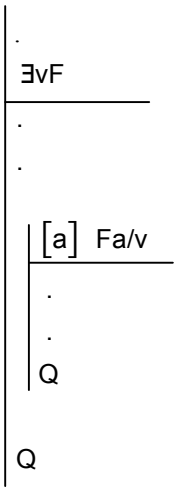
For example, ∧-Elim gives us the following one-step proof:



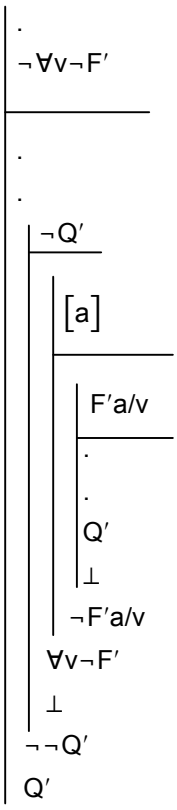
But using just the ¬, → and ⊥ rules, we can give the following proof of the "translation" of this argument. A and B might themselves be sentences containing ∧, ∨ and ∃; so let A′ and B′ be their translations, respectively.



For another example, suppose ∃-Elim gives us the following proof:



On the assumption that Q can be derived from Fa/v, we can do the "same thing" without ∃-Elim using our remaining rules. Suppose that F′ is the translation of F and Q′ is the translation of Q. We can assume that if Q can be derived from Fa/v, then Q′ can be derived from F′a/v (because we're thinking of this as a step in an inductive proof on the complexity of sentences).



Exercise: Show that we can similarly "prove" ∨-Elim using only the rules that are left after we throw away the ∧, ∨ and ∃.

Some Conventions and Definitions

Notational conventions: Upper case sans-serif letters from before the middle of the alphabet used as predicates (F, G, etc.) will be predicates of the object language. Upper case sans-serif letters from after the middle of the alphabet used as predicates (P, Q, etc.) will be schematic predicate letters of the metalanguage. Single upper-case sans-serif letters from the beginning of the alphabet (A, B, etc.) will be abbreviations of sentences of the object language. Single upper-case sans-serif letters from the beginning of the alphabet (P, Q, etc.) will be schematic sentence letters of the metalanguage.

Again, ⊥ is an abbreviation of a particular contradiction, let's say ¬(Fa → Fa).

We will not say that a sentence "follows" from some other sentences, since that is ambiguous. We will say either that the sentence can be *derived* from those other sentences using our proof rules, or we will say that the sentence is a *first-order consequence* of those other sentences.

Where A is a (possibly infinite) set of sentences of language $\mathcal{L}$ and P is a sentence of $\mathcal{L}$, we say that $A \vdash$ P if and only if there is a *proof in* $\mathcal{L}$ of P from A. (Alternatively, we say P can be *derived* from A; and P is a *syntactic consequence of* A.) "⊢" denotes the *syntactic consequence relation for* $\mathcal{L}$. The symbol "⊢" is called the *single turnstile*.

Where A is a (possibly infinite) set of sentences of $\mathcal{L}$, and P is a sentence of $\mathcal{L}$,, we say that $A \models$ P if and only if P is a first-order consequence in $\mathcal{L}$ of A. (Alternatively, we say the argument having the sentences in A as premises and P as conclusion is first-order valid; and P is a (first-order) *semantic consequence of* A.) "⊨" is the *double turnstile.*

We will define proofs and subproofs as certain kinds of sequences whose members are sentences and other sequences. When I speak of a *member* of a sequence, such as a sentence or another sequence, I do *not* include sentences or sequences that are members of members. So the sequence ⟨A, ⟨B, C⟩⟩ has just two members, A and ⟨B, C⟩. B, for instance, is not a member of the sequence ⟨A, ⟨B, C⟩⟩. So also, in counting sentences in a sequence, we do not count the sentences in a subproof in the sequence.

One sequence *S′* is a *component* of another sequence *S* if and only if either *S′* is identical to *S* or *S* is a member of *S* or *S′* is a member of a component of *S*. So the components of ⟨D, E, ⟨A, ⟨B, C⟩⟩⟩ are ⟨D, E, ⟨A, ⟨B, C⟩⟩⟩ itself, ⟨A, ⟨B, C⟩⟩ and ⟨B, C⟩. (B and C are not "components".)

Proofs as Sequences

We have been thinking of proofs as things that we write on the vertical dimension with "bars" marking subproofs. Now we are going to start thinking of proofs as sequences. The first member of a sequence that is a proof will be a set, the set of premises. Subsequenct members will be sentences or other sequences. The sequences that are components of the sequences that are proofs will be subproofs.

All of our proof rules can be reconceived as "permissions" on such sequences. For example: The rule →Intro can be written:

⟨…, ⟨P, …, Q⟩, …, (P → Q), ….⟩

In a sequence of this form, we will say that (P → Q) can be derived by →-Intro from ⟨P, …, Q⟩.

The rule ∀-Intro can be written:

⟨…, ⟨n, …, Pn/v⟩, …, ∀vP, …⟩, where n does not occur at any higher point in the sequence and does not occur in P.

Examples:

⟨{(A → B), (B → C)}, ⟨A, B, C⟩, (A → C)⟩ is a proof. We will say that it has *depth 1* because the only sequence that it contains contains no further sequence. We will say that A and (A → B) are "higher" than B and that B can be derived from higher lines by →-Elim. Similarly, C can be derived from higher lines by →-Elim. (A → C) can be derived from ⟨A, B, C⟩ by →-Intro.

⟨{(A → B)}, ⟨¬B, ⟨A, B, ⊥⟩,¬A⟩ (¬B → ¬A)⟩ is a proof. It has *depth 2* because it contains a sequence that contains a sequence (but that sequence contains no further sequence). The sentences that are higher than B are A, ¬B, and (A→ B). However, A is not higher than ¬A and ¬A is not higher than (¬B → ¬A).

⟨{∀xF(x), ∀x(F(x) → G(x))}, ⟨a, F(a), (F(a) → G(a), G(a)⟩, ∀xG(x)⟩ is a proof of depth 1.

(Notice that I do not require that the sentence justified by the subproof immediately follow the subproof. Fitch does not require that either, although I never pointed that out.)

The rules ¬-Elim, →-Elim, ⊥-Intro, ⊥-Elim, or ∀-Elim, Reit are our *inference rules.*

The rules ¬-Intro, →-Intro, and ∀-Intro are our *structural rules.* (These are also called *inference* rules in a *broad* sense of the term.)

The definition of proof

So what is a proof? We still lack a precise definition of proofs as sequences. (I never did give you a precise definition of a proof when we were writing proofs vertically.) We define this concept inductively. Throughout I assume we are talking about the sentences of a particular language $\mathcal{L}$, although I will usually suppress reference to it.

A *declaration:* A name (treated as an assumption for ∀-Intro).

A *prima facie subproof of depth 0:* A (finite) sequence whose first member is either a sentence (an assumption) or a declaration and whose subsequent members are all sentences.

A *prima facie subproof of depth* m+1: A (finite) sequence whose last member is a sentence (the conclusion), whose first member is either a sentence (an assumption) or a declaration and whose subsequent members are all either sentences or prima facie subproofs of depth no greater than m, and which contains at least one subproof of depth m.

A *prima facie proof of depth 0:* A (finite) sequence whose first member is a (possibly infinite) set of sentences (the premises) and whose subsequent members are all sentences.

A *prima facie proof of depth* m+1: A finite sequence whose last member is a sentence (the conclusion), whose first member is a set of sentences (the premises) and whose subsequent members are all either sentences or prima facie subproofs of depth no greater than m, and which contains at least one subproof of depth m.

An *item* is either a sentence or a prima facie subproof.

If *i* is an item and Q is a sentence and *i* and Q are members of components of sequence *S*, then *i is higher than* Q *in S* if and only if either (i) *i* is a member of the set of premises and Q is not, or (ii) *i* and Q are members of the sequence *S′*, which is a component of *S*, and *i* is earlier than Q in *S′*, or (iii) *i* is a member of a sequence *S′*, a component of *S*, and sequence *S″* is a member of *S′*, and *i* is earlier than *S″* in *S′*, and Q is a member of a component of *S″*.

A *subproof S′ of depth 0 in prima facie subproof or proof S*: A prima facie subproof of depth 0 that is a component (not necessarily a member) of a prima facie subproof or prima facie proof *S* of depth greater than 0 such that each member (sentence) of *S′* either (i) is an assumption or a declaration at the beginning of *S* or (ii) can be derived by one of our inference rules from sentences that are higher in *S* than it.

A *subproof S′ of depth m+1 in prima facie subproof or proof S*: A prima facie subproof of depth *m*+1 that is a component (not necessarily a member) of a prima facie subproof or proof *S* of depth greater than *m*+1 such that each item in *S′* either (i) is an assumption or a declaration at the beginning of *S′* or (ii) can be derived by one of our inference rules or structural rules from items that are higher in *S* than it or (iii) is a subproof in *S′* of depth no greater than *m* in *S*, and there is at least one subproof in *S′* that has a depth of *m*.

A *proof of depth 0:* A prima facie proof of depth 0 whose first member is a set of sentences (the premises) and whose subsequent members are all sentences that can be derived from earlier members by one of our inference rules.

A *proof S of depth m*+1: A prima facie proof of depth *m* + 1 whose members subsequent to the set of premises are all either (i) sentences that can be derived from earlier items by one of our rules, or (ii) subproofs in *S* of depth no greater than *m*, and there is at least one subproof in *S′* has a depth of *m*.

A *proof* is a sequence that for some $m \geq 0$ is a proof of depth *m*.

Notice that every subproof is also a prima facie subproof, and every proof is also a prima facie proof.

We will say that *there is a proof of* P from *A* if and only if there is a *proof* for which the premises are the sentences in *A* and the last item is the sentence P.

Some preliminary lemmas:

I make the following assumptions about first-order consequence (provable from the definition of first-order consequence):

Semantic Weakening: If $A \models \mathsf{P}$ and $A \subseteq B$, then $B \models \mathsf{P}$.
Semantic Cut: If $A \models \mathsf{P}$ and for all $\mathsf{Q} \in A$, $B \models \mathsf{Q}$, then $B \models \mathsf{P}$.
(Semantic Weakening is an immediate consequence of Semantic Cut. I will make frequent use of Weakening without mentioning it.)

Lemma 1: (a) Every argument in which the conclusion can be derived from the premises by one of our inference rules is first-order valid, and (b) each of our structural rules is validity-preserving.

Proof of (a): Exercise. Use the definition of first-order validity.

Proof of (b):

¬-Intro: The claim is that if $A \cup \{\mathsf{Q}\} \models \bot$, then $A \models \neg\mathsf{Q}$. Suppose $A \not\models \neg\mathsf{Q}$. Then there is a structure $\mathfrak{M}$ *such* that every sentence in A is true in $\mathfrak{M}$ and $\neg\mathsf{Q}$ is not true in $\mathfrak{M}$. So every sentence in A is true in $\mathfrak{M}$ and Q is true in $\mathfrak{M}$. But $\bot$ is not true in $\mathfrak{M}$. So $A \cup \{\mathsf{Q}\} \not\models \bot$.

→-Intro: If $A \cup \{\mathsf{P}\} \models \mathsf{Q}$, then $A \models (\mathsf{P} \rightarrow \mathsf{Q})$. Exercise.

∀-Intro: The claim is that if $A \models \mathsf{Pn/v}$, where n is a name that does not occur in any member of A and does not occur in P, then $A \models \forall\mathsf{vP}$. Suppose $A \not\models \forall\mathsf{vP}$. Then there is a structure $\mathfrak{M}$ and an object o in the domain of the structure such that every sentence in A is true in the structure but $g_{\varnothing}(\mathsf{v}/o)$ does not satisfy P. Now consider a structure $\mathfrak{M}'$ just like $\mathfrak{M}$ except that $\Sigma(\mathsf{n}) = o$. A sentence that does not contain n is true in $\mathfrak{M}'$ if and only if it is true in $\mathfrak{M}$. (Strictly speaking, we should prove that by induction on the complexity of sentences; but it's obvious.) Likewise, if n does not occur in P, then $g_{\varnothing}$ satisfies $\mathsf{Pn/v}$ in $\mathfrak{M}'$ ($\mathsf{Pn/v}$ is true in $\mathfrak{M}'$) if and only if $g_{\varnothing}(\mathsf{v}/o)$ satisfies P in $\mathfrak{M}$. So every sentence in A is true in $\mathfrak{M}'$, but $\mathsf{Pn/v}$ is not true in $\mathfrak{M}'$. So $A \not\models \mathsf{Pn/v}$.

Lemma 2: Let S' be a subproof of depth 0 in proof S. Each sentence after the assumption (if there is one) is a first-order consequence of the assumption together with sentences higher in the proof.

Proof: By induction.

Basis: Show that the first sentence after the assumption has this property. The first sentence can be derived from the assumption and higher sentences by one of our inference rules. So the thesis holds by Lemma 1 (a).

Induction Hypothesis: The first through *n*th sentences after the assumption of *S′* are first-order consequences of the assumption and sentences higher than *S′* in *S*.

Induction Step: We have to show that the *n*+1st sentence after the assumption is a first-order consequence of the assumption and sentences higher than *S′* in *S*. This is an immediate consequence of the induction hypothesis and Semantic Cut. (In the statement of Semantic Cut, let *B* consist of the assumption and higher sentences, and let *A* consist of the assumption, higher sentences and the first through *n*th sentences after the assumption.)

Lemma 3: Let *S′* be a subproof in proof *S*. Suppose that for every subproof *S″* in *S′*, every sentence in *S″* after the assumption (if there is one) is a first-order consequence of the assumption of *S″* and sentences in *S* higher than *S″* (which includes sentences higher than *S″* in *S′*). Then every sentence in *S′* after the assumption is a first-order consequence of the assumption of *S′* and sentences higher than *S′* in *S*.

Basis: Show that the first sentence P in *S′* after the assumption or declaration is a first-order consequence of the assumption and sentences higher in *S*. We have two cases to consider:

Case 1: P can be derived by an inference rule from the assumption of *S′* and sentences higher in *S*. Then the thesis is a consequence of Lemma 1 (a).

Case 2: P can be derived by a structural rule from a subproof *S″* in *S′*. (So the first item in *S′* after the assumption or declaration is not a sentence but the subproof *S″*.) Then by the hypothesis of the lemma and Lemma 1 (b), P is a first-order consequence of the assumption of *S′* and sentences higher than *S′* in *S*.

(For example: Lemma 1 (b) tells us that if $A \cup \{Q\} \models \bot$, then $A \models \neg Q$. Think of *A* as comprising the assumption of *S′* and the sentences higher than *S′* in *S*, think of Q as the assumption of *S″*, think of ⊥ as the last item in *S″*, and think of ¬Q as P.)

Induction Hypothesis: Each of the first through *n*th sentences after the assumption or declaration in *S′* are first-order consequences of the assumption of *S′* and higher items in *S*.

Induction Step: Where P is the *n*+1st sentence after the assumption in *S′*, we have to show that P is a first-order consequence of the assumption of *S′* and sentences higher than *S′* in *S*.

Case 1: P can be derived by an inference rule from sentences earlier in *S′* and higher than *S′* in *S*. By Lemma 1 (a), P is a first-order consequence of sentences earlier in *S′* and higher than *S′* in *S*. So by the induction hypothesis and Cut, P is a first-order consequence of the assumption of *S′* and sentences higher than *S′* in *S*.

Case 2: P can be derived by a structural rule from a subproof *S″* in *S′*. By Lemma 1 (b), P is a first-order consequence of sentences higher than *S″* in *S*. Then by the induction hypothesis and Cut, P is a first-order consequence of the assumption of *S′* and sentences higher than *S′* in *S*.

Lemma 4: In any subproof *S′* in any proof *S*, every sentence in *S′* after the assumption has the property of being a first-order consequence of the assumption of *S′* and sentences higher than *S′* in *S*.

Basis: Let *S′* be a subproof of depth 0 in proof *S*. By Lemma 2, *S* has the property.

Induction Hypothesis: Any subproof of depth no greater than *m* in *S* has the property.

Induction Step: We need to show that any subproof of depth *m*+1 has the property. This is an immediate consequence of Lemma 3.

The Soundness Theorem for First-order Logic: If $A \vdash Q$, then $A \models Q$.

Proof: We prove something stronger: Each sentence in a proof is a first-order consequence of the premises. Again, we proceed by induction. (The proof is similar to the proof of Lemma 3.)

Basis: Let *S* be a proof. Where P is the first sentence in *S* after the premises, show that P is a first-order consequence of the premises. Two cases:

Case 1: P can be derived from the premises by an inference rule. By Lemma 1 (a), P is a first-order consequence of the premises.

Case 2: P can be derived from a subproof by a structural rule. (So the first item after the premises is a subproof.) By Lemma 4, the last sentence of the subproof is a first-order consequence of the assumption and the premises (which are the only higher sentences). So by Lemma 1 (b), P is a first-order consequence of the premises.

Induction Hypothesis: The first through *n*th lines after the premises are first-order consequences of the premises.

Induction Step: Where P is the *n*+1st line, we have to show that P is a first-order consequence of the premises.

Case 1: P can be derived from earlier sentences by an inference rule. By Lemma 1 (a), P is first-order consequence of the earlier sentences. Then the thesis holds by the induction hypothesis and Cut.

Case 2: P can be derived from a subproof *S′* in *S* by a structural rule. By Lemma 4, the last sentence in *S′* is a first-order consequence of the assumption of *S′* and sentences higher than *S′* in *S*. So by Lemma 1 (b), P is a first-order consequence of sentences higher than *S′* in *S*. So by the induction hypothesis and Cut, P is a first-order consequence of the premises.

Lesson 3: The Completeness Theorem for Truth-functional Logic

Recall the Soundness Theorem: If $A \vdash Q$, then $A \models Q$.

We now want to prove the Completeness Theorem (for first-order logic):
If $A \models Q$, then $A \vdash Q$.

However, we will approach it in stages.

Define: $A \vdash_{tt} Q$ if and only if Q can be derived from sentences in A using only the introduction and elimination rules for $\neg$, $\rightarrow$, and $\perp$-Intro. (In other words, there is a proof, call it *tt-proof*, of Q from A using just those rules.) Call these the *sentential* rules (since they do not deal in subformulas). I assume that the relation $\vdash_{tt}$ pertains to a particular language, although I usually suppress reference to it.

We do not need $\perp$-Elim, because if we can construct a proof like $\langle \ldots, \perp, Q \rangle$, then we can construct a proof like $\langle \ldots, \langle \neg Q, \ldots, \perp \rangle, \neg\neg Q, Q \rangle$. We also do not need Reit, because if we can construct a proof like $\langle \ldots, P, \ldots, P \rangle$, then we can construct a proof like $\langle \ldots, P, \ldots, \langle \neg P, \perp \rangle, \neg\neg P, P \rangle$.

Let *val* (lower case "*v*") be an assignment of the truth values T and F to the atomic and quantified sentences (i.e., noncompound sentences) of $\mathcal{L}$.

Let *Val* be a function from the sentences of $\mathcal{L}$ into {T, F} such that
Val(P) = T (capital "*V*") if and only if either:
(i) P is an atomic or quantified sentence and *val*(P) = T, or
(ii) P = $\neg$Q and *Val*(Q) = F, or
(iii) P = (Q $\rightarrow$ R) and either *Val*(Q) = F or *Val*(R) = T.

(Remember that we have thrown out $\wedge$, $\vee$ and $\leftrightarrow$.)

So *Val*, called an *evaluation*, "extends" to the rest of the language the truth value assignment *val*.

Define: $A \models_{tt} Q$ if and only if for every truth value assignment *val*, if for all $P \in A$, *Val*(P) = T, then *Val*(Q) = T. (In other words, Q is a tautological consequence of A. Again, I assume we are talking about a particular language, although I suppress reference to it.)

Define: A is *tt-satisfiable* if and only if there is an assignment of truth values to the atomic and quantified components *val* such that every member of A is true on that assignment, i.e., for all $\mathsf{S} \in A$, $Val(\mathsf{S}) = \mathsf{T}$. ($A$ is truth-functionally consistent.)

Before we prove the Completeness Theorem, we will prove the completeness of the sentential rules relative to tautological consequence:

The Truth-functional Completeness Theorem: If $A \models_{tt} \mathsf{Q}$, then $A \vdash_{tt} \mathsf{Q}$.

Here is how we will do that. We will prove the following three Lemmas, from which the Truth-functional Completeness Theorem follows:

Lemma 1: If $A \nvdash_{tt} \mathsf{S}$ then $A \cup \{\neg \mathsf{S}\} \nvdash_{tt} \bot$.

Lemma 2: If $B \nvdash_{tt} \bot$, then B is tt-satisfiable.

Lemma 3: If $A \cup \{\neg \mathsf{S}\}$ is tt-satisfiable, then $A \not\models_{tt} \mathsf{S}$.

The Truth-functional Completeness Theorem immediately follows from L1-L3:

If $A \nvdash_{tt} \mathsf{S}$ then $A \not\models_{tt} \mathsf{S}$.

i.e., if $A \models_{tt} \mathsf{S}$, then $A \vdash_{tt} \mathsf{S}$.

(Think of B in L2 as $A \cup \{\neg \mathsf{S}\}$.) The hard part will be to prove Lemma 2.

Proof of Lemma 1:

We will prove it in this form: If $A \cup \{\neg \mathsf{S}\} \vdash_{tt} \bot$, then $A \vdash_{tt} \mathsf{S}$.

Observe: If we can have a proof like this:

$\langle A \cup \{\neg \mathsf{S}\}, \ldots, \bot \rangle$

then we can have a proof like this:

$\langle A, \langle \neg \mathsf{S}, \ldots, \bot \rangle \rangle$.

Add two steps, by ¬-Intro and ¬Elim.

$\langle A, \langle \neg \mathsf{S}, \ldots, \bot \rangle, \neg\neg \mathsf{S}, \mathsf{S} \rangle$.

Thus, we obtain a proof of S from A.

Proof of Lemma 3:

If there is a truth value assignment *val* such that for all $\mathsf{P} \in A$, $Val(\mathsf{P}) = \mathsf{T}$ and $Val(\neg \mathsf{S}) = \mathsf{T}$, then there is a truth value assignment, namely, the same one, such that for all $\mathsf{P} \in A$, $Val(\mathsf{P}) = \mathsf{T}$ and $Val(\mathsf{S}) = \mathsf{F}$.

Lemma 2 is an immediate consequence of the following two theorems:

Theorem: Satisfiability of formally consistent and complete sets:

Suppose that M is a set of sentences that is formally consistent and formally complete, that is:
(i) $M \nvdash_{tt} \bot$ (formal consistency), and
(ii) For all sentences S, either $M \vdash_{tt} \mathsf{S}$ or $M \vdash_{tt} \neg \mathsf{S}$ (formal completeness).
Then there is a truth value assignment *val* such that for all $\mathsf{S} \in M$, $Val(\mathsf{S}) = \mathsf{T}$.

Theorem: Completability of formally consistent sets:

Suppose B is formally consistent (i.e., $B \nvdash_{tt} \bot$). Then there is a formally consistent, formally complete set M such that $B \subseteq M$.

Proof of Lemma 2, given these theorems:

Suppose that $B \nvdash_{tt} \bot$. By the completability of formally consistent sets, there is a formally consistent, formally complete set M such that $B \subseteq M$. By the satisfiability of formally consistent and complete sets, M is tt-satisfiable. Since $B \subseteq M$, B is tt-satisfiable too.

It remains to prove the satisfiability of formally consistent and complete sets and the completability of formally consistent sets.

Lemma 4 (= Lemma 3 in Barwise and Etchemendy, p. 472)**:**
Suppose A is a formally consistent and formally complete set of sentences.
1. $A \vdash_{tt} \neg P$ if and only if $A \nvdash_{tt} P$.
2. $A \vdash_{tt} (P \rightarrow Q)$ if and only if either $A \nvdash_{tt} P$ or $A \vdash_{tt} Q$.

(Note: You would expect a proof of the completeness of the sentential proof rules to *use* those rules somehow. They are used here, in proving Lemma 4, as well as in the proof of the completability of formally consistent sets.)

Proof of 1:

Left-to-right: By the assumption that A is formally consistent, if $A \vdash \neg P$, then $A \nvdash P$.

Right-to-left: By the assumption that A is formally complete, if $A \nvdash P$, then $A \vdash \neg P$.

Proof of 2:

Right-to-left:

Case (i): Suppose $A \nvdash_{tt} P$. Since A is formally complete, $A \vdash_{tt} \neg P$. So we can construct a proof like the following, using $\perp$-Intro, $\neg$-Intro, $\neg$-Elim and $\rightarrow$-Intro: $\langle A, \ldots, \neg P, \langle P, \langle \neg Q, \perp \rangle, \neg\neg Q, Q \rangle, (P \rightarrow Q) \rangle$.

Case (ii): Suppose $A \vdash_{tt} Q$. Then we can use $\rightarrow$-Intro to constuct a proof like this: $\langle A, \langle P, \ldots, Q \rangle, (P \rightarrow Q) \rangle$.

Left-to-right:

Suppose $A \vdash_{tt} P$ and $A \nvdash_{tt} Q$. Since A is complete, $A \vdash_{tt} \neg Q$.

So we can construct the following proof, using $\rightarrow$-Elim, $\perp$-Intro, and $\neg$-Intro: $\langle A, \ldots, P, \neg Q, \langle (P \rightarrow Q), Q, \perp \rangle, \neg (P \rightarrow Q) \rangle$.

Since A is formally consistent, $A \nvdash_{tt} (P \rightarrow Q)$.

Proof of the satisfiability of formally consistent and formally complete sets:

Suppose M is formally consistent and complete. Let *val* be such that for all atomic and quantified sentences S of $\mathcal{L}$, *val*$(S) = T$ if and only if $M \vdash_{tt} S$. We prove by induction that for *all* sentences S of $\mathcal{L}$, *Val*$(S) = T$ if and only if $M \vdash_{tt} S$. In that case, for all $S \in M$, *Val*$(S) = T$.

Basis: The thesis holds for all atomic and quantified sentences, by the definition of *Val*.

Induction hypothesis: Suppose the thesis holds for arbitrary sentences Q and R of $\mathcal{L}$.

Induction step:

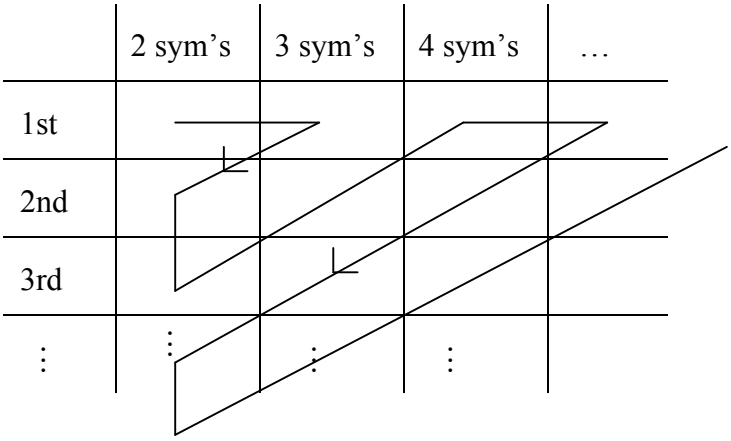
(¬) Suppose P = ¬Q. *Val*(P) = T iff *Val*(Q) = F iff (by IH) $M \nvdash_{tt}$ Q iff (by Lemma 4) $M \vdash_{tt} \neg$Q.

(→) Suppose P = (Q → R). *Val*(P) = T iff *Val*(Q) = F or *Val*(R) = T iff (by IH) $M \nvdash_{tt}$ Q or $M \vdash_{tt}$ R iff (by Lemma 4) $M \vdash_{tt}$ (Q → R).

But for all S ∈ *M*, $M \vdash_{tt}$ S. So for all S ∈ *M*, *Val*(S) = T, i.e., *M* is tt-satisfiable.

Enumerating the sentences of ℒ.

We will need to assume that there is an infinite list of the sentences of ℒ. Since there are infinitely many names, variables and predicates in the language, we cannot expect to list them in alphabetical order. Here is how we can do it. Suppose we have an infinite list of the sentences that are two symbols long (e.g., Fa, Fb, etc.), and an infinite list of the sentences that are three symbols long, and so on. So we have an infinite number of infinite lists. Arrange the lists in a table, with each list occupying one column, thus:



And then, produce a single list of *all* sentences by following the zig-zag line through the table. (This is called *zig-zagging* through the table.)

But now, how do we produce the list of two-symbol sentences? Here's how: Construct a table with a list of predicates running down the left and a list of names running across the top, and in each cell write the predicate followed by the name, thus:

	a	b	c	…
A	Aa	Ab	Ac	…
B	Ba	Bb	Bc	…
C	Ca	Cb	Cc	…
⋮	⋮	⋮	⋮	

Finally, produce the list of two-symbol sentences by zig-zagging through the table.

Exercise: Describe a method for listing the three-symbol sentences. (Hint: Think "three dimensions".)

Note: When it comes to listing, say, the seven-symbol sentences, the easiest thing might be just to describe a method for listing all *strings* of seven symbols of $\mathcal{L}$ and then say, "Go through the list of seven-symbol strings and add to the list of seven-symbol sentences each of those that happens to be a sentence of $\mathcal{L}$."

One more note about constructing such lists: If we really want to imagine generating the list of sentences, we cannot suppose that we are "given" a bunch of tables each of which has infinitely long columns and infinitely long rows. Rather, we have to imagine we have a set of instructions that allows us to add a cell to each of the tables we are using as we need it.

Proof of completability of formally consistent sets.

Let B be formally consistent.
Let $\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2, \ldots$ be an enumeration of all atomic and quantified sentences (which can be produced in some such manner as that just described).

Define M thus:
Let $B_0 = B$.
For each $i \geq 0$, let $B_{i+1} = B_i \cup \{\mathsf{A}_i\}$ if $B_i \cup \{\mathsf{A}_i\} \nvdash_{\mathrm{tt}} \bot$.
Otherwise, let $B_{i+1} = B_i$.
Let $M = B_0 \cup B_1 \cup B_2 \cup \ldots = \bigcup_{i=0}^{\infty} B_i$. (Clearly, $B \subseteq M$.)
(In other words, $\mathsf{S} \in M$ if and only if $\mathsf{S} \in B_0$ or $\mathsf{S} \in B_1$ or $\mathsf{S} \in B_2$ or … .)

M is formally consistent.
Suppose not. That is, $M \vdash_{tt} \bot$.
Since proofs are finite, there is a smallest j such that $B_j \vdash_{tt} \bot$.
But by the construction, there is no such j.
(The fact that proofs are finite means that only finitely many members of M are used in the proof. So we can pick the smallest j such that B_j includes them all.)

M is formally complete.
That is to say, for all sentences S in $\mathcal{L}$, $M \vdash_{tt} \mathsf{S}$ or $M \vdash_{tt} \neg\mathsf{S}$.
We prove this by induction.

Basis:
Suppose A_j is an atomic or quantified sentence and j is its place in the enumeration.
Suppose $M \nvdash_{tt} \mathsf{A}_j$. In that case, by the construction of M, $B_j \cup \{\mathsf{A}_j\} \vdash_{tt} \bot$.
So by $\neg$-Intro, $B_j \vdash_{tt} \neg\mathsf{A}_j$. So $M \vdash_{tt} \neg\mathsf{A}_j$.
So either $M \vdash_{tt} \mathsf{A}_j$ or $M \vdash_{tt} \neg\mathsf{A}_j$.

Induction hypothesis: Suppose for arbitrary Q and R of $\mathcal{L}$, $M \vdash_{tt} \mathsf{Q}$ or $M \vdash_{tt} \neg\mathsf{Q}$ and $M \vdash_{tt} \mathsf{R}$ or $M \vdash_{tt} \neg\mathsf{R}$.

Induction step:
($\neg$) Suppose $\mathsf{P} = \neg\mathsf{Q}$.
By the induction hypothesis, $M \vdash_{tt} \mathsf{Q}$ or $M \vdash_{tt} \neg\mathsf{Q}$.
Case (i): $M \vdash_{tt} \neg\mathsf{Q}$.
Case (ii): $M \vdash_{tt} \mathsf{Q}$. So we have a *proof* like $\langle M, \ldots, \mathsf{Q}\rangle$.
Using $\bot$-Intro, $\neg$-Intro and $\neg$-Elim, we can construct a proof like $\langle M, \langle \neg\mathsf{Q}, \ldots, \mathsf{Q}, \bot\rangle, \neg\neg\mathsf{Q}\rangle$.
So $M \vdash_{tt} \neg\neg\mathsf{Q}$.
So in both cases, we have either $M \vdash_{tt} \mathsf{P}$ or $M \vdash_{tt} \neg\mathsf{P}$.
($\rightarrow$) Suppose $\mathsf{P} = (\mathsf{Q} \rightarrow \mathsf{R})$.
Case (i): $M \vdash_{tt} \neg\mathsf{Q}$. In that case, we can construct a proof like this: $\langle M, \ldots, \neg\mathsf{Q}, \langle \mathsf{Q}, \langle \neg\mathsf{R}, \bot\rangle, \neg\neg\mathsf{R}, \mathsf{R}\rangle, (\mathsf{Q} \rightarrow \mathsf{R})\rangle$.
Case (ii): $M \vdash_{tt} \mathsf{R}$. In that case, we can construct a proof like this: $\langle M, \langle \mathsf{Q}, \ldots, \mathsf{R}\rangle, (\mathsf{Q} \rightarrow \mathsf{R})\rangle$.
Case (iii): By the induction hypothesis, if cases (i) and (ii) do not hold, then $M \vdash_{tt} \mathsf{Q}$ and $M \vdash_{tt} \neg\mathsf{R}$. In that case, we can construct a proof like this: $\langle M, \langle (\mathsf{Q} \rightarrow \mathsf{R}), \ldots, \mathsf{Q}, \mathsf{R}, \ldots, \neg\mathsf{R}, \bot\rangle, \neg(\mathsf{Q} \rightarrow \mathsf{R})\rangle$.
So in all cases, we have either $M \vdash_{tt} (\mathsf{Q} \rightarrow \mathsf{R})$ or $M \vdash_{tt} \neg(\mathsf{Q} \rightarrow \mathsf{R})$.

This completes the proof of the Truth-functional Completeness Theorem.

Lesson 4: The Completeness Theorem for First-order Logic

Now we want to prove the completeness theorem for first-order logic:

If $A \models \mathsf{Q}$, then $A \vdash \mathsf{Q}$.

Recall that $A \vdash \mathsf{Q}$ means that where A is a set of sentences in the language $\mathcal{L}$ and Q is a sentence in the language $\mathcal{L}$, there is a proof of Q from the sentences in A (using any of our introduction and elimination rules). Here we are thinking of $\mathcal{L}$ as a specific first-order language. When we need to be specific about the language, we will write, $A \vdash_{\mathcal{L}} \mathsf{Q}$.

We have already proved that if $A \models_{tt} \mathsf{Q}$, then $A \vdash_{tt} \mathsf{Q}$. So what we want to do now is "extend" that result from truth-functional validity to first-order validity.

Let $\mathcal{L}$+ be a language just like $\mathcal{L}$ except that it contains denumerably many additional individual constants beyond those that $\mathcal{L}$ contains. (We will also call individual constants *names* or just *constants.*)

So $A \vdash_{\mathcal{L}+} \mathsf{Q}$ will mean that where A is a set of sentences in $\mathcal{L}$+ and Q is a sentence in $\mathcal{L}$+, Q can be derived from sentences in A using our introduction and elimination rules (i.e., there is a proof in $\mathcal{L}$+ using those rules).

Where P is a well-formed formula of a first order language $\mathcal{L}$, and n is an individual constant of $\mathcal{L}$, and v is a variable of $\mathcal{L}$, $\mathsf{Pn/v}$, as before (in Lecture 2), is the result of substituting n for v wherever v occurs free in P. For example, $\exists \mathsf{xRxy\ a/y = \exists xRxa}$.

Outline of proof:

We will show how to construct a set of sentences H (called the *Henkin theory*) in the language $\mathcal{L}$+ that meets the following conditions:

(i) *The Elimination Theorem:*

Where A is a set of sentences of $\mathcal{L}$ (the original language) and Q is a sentence of $\mathcal{L}$, if $A \cup H \vdash_{\mathcal{L}+} \mathsf{Q}$, then $A \vdash \mathsf{Q}$ (i.e., $A \vdash_{\mathcal{L}} \mathsf{Q}$).

(In other words, everything that we can prove (in $\mathcal{L}$) with the help of H (in $\mathcal{L}$+), we can prove without it. Only the members of H contain the extra constants of $\mathcal{L}$+.)

(ii) *The Henkin Construction Theorem:*

For every truth value assignment *val*, if for all $\mathsf{P} \in H$, $Val(\mathsf{P}) = \mathsf{T}$, then there is a structure $\mathfrak{M}_{val}$ such that for all P of $\mathcal{L}$+, if $Val(\mathsf{P}) = \mathsf{T}$, then P is true in $\mathfrak{M}_{val}$.

(In other words, if a truth value assignment *Val* assigns truth to every member of *H*, then there is a first-order structure that assigns truth to every sentence that *Val* assigns truth to. I write "$\mathfrak{M}_{val}$" as a mnemonic, tell help you remember that it's the structure that *corresponds* to *val*.)

Suppose we can establish the existence of such an *H*. The completeness theorem can then be proved as follows:

Proof of completeness given a Henkin theory:

Suppose $A \models \mathsf{Q}$, where *A* is a set of sentences of $\mathcal{L}$ and Q is a sentence of $\mathcal{L}$.
Then there is no structure $\mathfrak{M}_{val}$ such that every member of $A \cup \{\neg \mathsf{Q}\}$ is true in $\mathfrak{M}_{val}$.
Then by the Henkin Construction Theorem, there is *no* truth value assignment *val* such that for every $\mathsf{P} \in A \cup H \cup \{\neg \mathsf{Q}\}$, $Val(\mathsf{P}) = \mathsf{T}$.
So $A \cup H \models_{\text{tt}} \mathsf{Q}$.
By the Truth-functional Completeness Theorem (applied to the language $\mathcal{L}$+),
$A \cup H \vdash_{\mathcal{L}+ \text{ tt}} \mathsf{Q}$.
So (since every proof in $\mathcal{L}$+ using just the sentential rules is also a proof in $\mathcal{L}$+),
$A \cup H \vdash_{\mathcal{L}+} \mathsf{Q}$.
So by the Elimination Theorem,
$A \vdash \mathsf{Q}$.

Some useful propositions:
(Assume that $\vdash$ is the syntactic consequence relation for an arbitrary first-order language.)

Proposition 1 (the Deduction Theorem):
If $A \cup \{\mathsf{P}\} \vdash \mathsf{Q}$, then $A \vdash (\mathsf{P} \rightarrow \mathsf{Q})$.
Proof: Exercise.

Proposition 2 (Syntactic Cut):
If (i) $A \cup \{\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_n\} \vdash \mathsf{Q}$ and (ii) for all *i*, $1 \leq i \leq n$, $A \vdash \mathsf{P}_i$, then $A \vdash \mathsf{Q}$.
(Compare the lemma called "Semantic Cut" in L2.)
Proof:
Suppose we have a proof like this:

$\langle A \cup \{P_1, P_2, \ldots, P_n\}, \ldots, Q\rangle$.
By Proposition 1, applied *n* times, we have a proof like this:
$\langle A, \ldots, (P_1 \to (P_2 \to (\ldots (P_n \to Q)\ldots)\rangle$.
By (ii), we have a proof like this:
$\langle A, \ldots, (P_1 \to (P_2 \to (\ldots (P_n \to Q)\ldots), P_1, P_2, \ldots, P_n\rangle$.
So by *n* applications of →Elim, we have a proof like this:
$\langle A, \ldots, Q\rangle$.

Proposition 3 (Lemma 7 in Barwise and Etchemendy, p. 536):
(1) If $A \vdash (P \to Q)$ and $A \vdash (\neg P \to Q)$, then $A \vdash Q$.
(2) If $A \vdash ((P \to Q) \to R)$, then $A \vdash (\neg P \to R)$ and $A \vdash (Q \to R)$.
Proof:
Part (1): We know $A \cup \{(P \to Q), (\neg P \to Q)\} \vdash Q$. So (1) follows by Prop. 2.
Part (2): Exercise.

Proposition 4 (comparable to Lemma 8 in Barwise and Etchemendy, p. 536):
Suppose n does not occur in P or any member of *A*.
Then if $A \vdash (\neg Pn/v \to Q)$, then $A \vdash (\neg \forall v P \to Q)$.
Proof: Note: This where we use ∀-Intro.
Suppose we have a proof like this:
$\langle A, \ldots, (\neg Pn/v \to Q)\rangle$.
Then we can construct a proof like this:
$\langle A, \langle \neg \forall v P, \langle \neg Q, \langle n, \langle \neg Pn/v, \ldots, (\neg Pn/v \to Q), Q, \bot\rangle, \neg\neg Pn/v, Pn/v\rangle, \forall v P, \bot\rangle, \neg\neg Q, Q\rangle, (\neg \forall v P \to Q)\rangle$.

Proposition 5 (compare Lemma 9 in Barwise and Etchemendy, p. 537):
Suppose n does not occur in P or any member of *A*.
Then if $A \cup \{(Pn/v \to \forall v P)\} \vdash Q$ then $A \vdash Q$.
Proof:
Suppose $A \cup \{(Pn/v \to \forall v P)\} \vdash Q$.
By Proposition 1, $A \vdash ((Pn/v \to \forall v P) \to Q)$.
By Proposition 3 part (2),
(i) $A \vdash (\neg Pn/v \to Q)$, and
(ii) $A \vdash (\forall v P \to Q)$.
From (i), by Proposition 4,
(iii) $A \vdash (\neg \forall v P \to Q)$.
From (ii) and (iii), by Proposition 3, part (1),
(iv) $A \vdash Q$.

Proposition 6 (compare Lemma 10 in Barwise and Etchemendy, p. 537):
$A \vdash (\forall v P \rightarrow Pn/v)$, and
$A \vdash ((Pn/v \wedge n = m) \rightarrow Pm/v)$, and
$A \vdash n = n$.

Proof: Exercise. This is where we use ∀-Elim and =-Elim and =-Intro.

Now we need to construct the Henkin theory. But first we need to construct the language of the Henkin theory, $\mathcal{L}+$. We will do this in stages. At each stage, we construct not only a new language, but also a *witness function* for that language.

Suppose we have a table of individual constants not in $\mathcal{L}$:

n_{00}	n_{10}	n_{20}	n_{30}	…
n_{01}	n_{11}	n_{21}	n_{31}	…
n_{02}	n_{12}	n_{22}	n_{32}	…
n_{03}	n_{13}	n_{23}	n_{33}	…
:	:	:	:	

In other words, each of the extra constants of $\mathcal{L}+$ should have place in this table so that we can identify it by its "double subscript". We will think of a language now as a *set* that includes all of the basic vocabulary and all of the wffs that can be built of from that vocabulary in accordance with the definition of a wff.

We further suppose that for any language we can produce an enumeration (a one-dimensional list) of all the formulas of that language containing exactly one free variable.

We now define a whole series of languages starting with $\mathcal{L}$ and culminating in $\mathcal{L}+$.

Let $\mathcal{L}_0 = \mathcal{L}$.

Let w_0 be a function such that if P_{0j} is the jth formula in an enumeration of the formulas of $\mathcal{L}_0$ having exactly one free variable, then $w_0(P_{0j}) = n_{0j}$.

For each $i \geq 0$, let
$\mathcal{L}_{i+1} = \mathcal{L}_i \cup \{w_i(P) \mid P \text{ is a wff of } \mathcal{L}_i \text{ with exactly one free variable}\}$.

(Notation: In general, $\{f(x) \mid \ldots x \ldots\}$ stands for the set of things that results from applying function f to the things that satisfy the condition $\ldots x \ldots$.)

For each $i \geq 0$, let

$$w_{i+1}(\mathsf{P}) = \begin{cases} w_i(\mathsf{P}) \text{ if } \mathsf{P} \text{ is a wff of } \mathcal{L}_i \\ \\ \mathsf{n}_{(i+1)j} \text{ if } \mathsf{P} \text{ is the } j\text{th wff in an enumeration of the wffs having exactly one free variable that are in } \mathcal{L}_{i+1} \text{ but not in } \mathcal{L}_i. \end{cases}$$

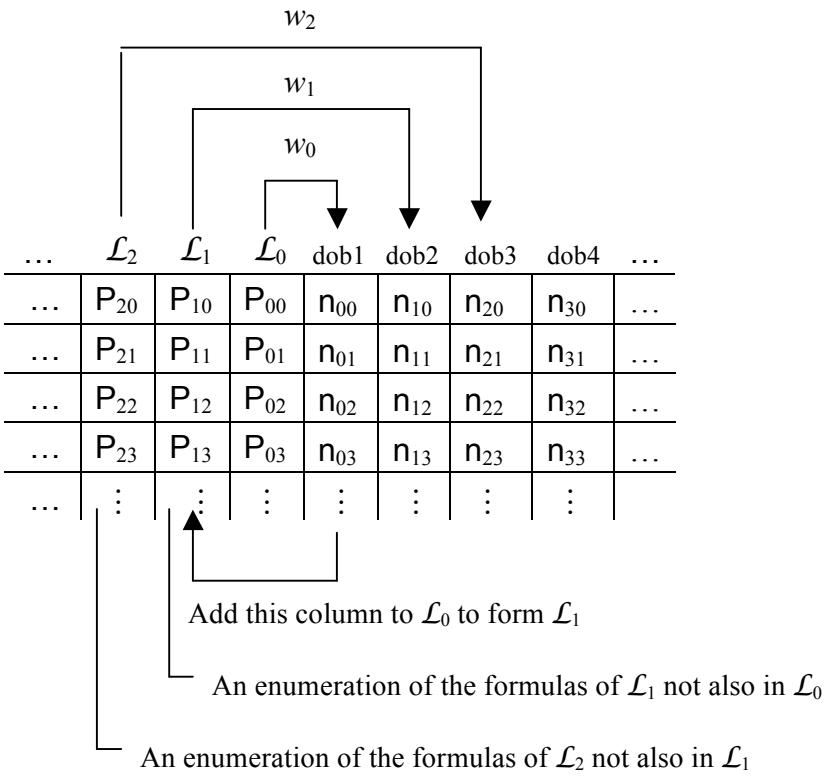
So the construction alternates between constructing the next language and constructing the next function. At each stage the sentences of the language constructed at that stage will contain all wffs that can be grammatically constructed using the witnesses for the formulas of every previous stage.

Let $\mathcal{L}+ = \bigcup_{i=0}^{\infty} \mathcal{L}_i$. (In other words, the union of $\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$, …)

Let $w = \bigcup_{i=0}^{\infty} w_i$.

(In defining a function through unions, we are thinking of functions as sets of ordered pairs.)

The situation can be illustrated in a diagram:



("dob" stands for *date of birth*.)

If $w(\mathsf{P}) = \mathsf{n}$, write c_P for n. So $w(\mathsf{P}) = \mathsf{c}_\mathsf{P}$. c_P is a *witness* for P. Notice that by our construction, we have ensured that for every formula of $\mathcal{L}$+ that has one free variable, that formula has such a witness.

We can *define* the Henkin theory H as follows:

A wff Q of $\mathcal{L}$+ is a member of H if and only if either:
(1) for some individual constant n, $\mathsf{Q} = \mathsf{n} = \mathsf{n}$, or
(2) P is a wff of $\mathcal{L}$+ containing at most v free, and either
  (a) for some constants m and n, $\mathsf{Q} = ((\mathsf{P}\mathsf{n}/\mathsf{v} \wedge \mathsf{n} = \mathsf{m}) \rightarrow \mathsf{P}\mathsf{m}/\mathsf{v})$, or
  (b) for some contant n, $\mathsf{Q} = (\forall\mathsf{v}\mathsf{P} \rightarrow \mathsf{P}\mathsf{n}/\mathsf{v})$, or
  (c) $\mathsf{Q} = (\mathsf{P}\mathsf{c}_\mathsf{P}/\mathsf{v} \rightarrow \forall\mathsf{v}\mathsf{P})$.

Notice the use of "c_P" in this definition. In the case where P does not contain v free, assume that $\mathsf{P}\mathsf{c}_\mathsf{P}/\mathsf{v} = \mathsf{P}$.

In the case where v is free in P, say that $(\mathsf{P}\mathsf{c}_\mathsf{P}/\mathsf{v} \rightarrow \forall\mathsf{v}\mathsf{P})$ is the *witnessing axiom* for c_P. (So where P does not contain v free, the witnessing axiom for P does not contain a witness.)

If n is a constant in $\mathcal{L}_0$ (=$\mathcal{L}$), then the *birth date of* n is 0.
If P is a formula in $\mathcal{L}_0$ and $w_0(\mathsf{P}) = \mathsf{n}$, say that the *birth date of* n is 1.
For $i > 0$, if P is a formula in $\mathcal{L}_i$ but not in $\mathcal{L}_{i-1}$, and $w_i(\mathsf{P}) = \mathsf{n}$, say that $i + 1$ is the *birth date of* n.

dob(n) = i if and only if the birth date of n is i.

In other words, the date of birth (*dob*) of a name is i iff $\mathcal{L}_i$ is the first language in the series containing formulas containing that name.

Observation:
If for some $i \geq 0$, *dob*(n) = $i + 1$, then n does not occur in any wff in any of $\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$, …, $\mathcal{L}_i$.

The Independence Lemma:

If c_P is not c_Q and *dob*(c_P) ≤ *dob*(c_Q), then c_Q is not in the witnessing axiom for c_P.

Proof:
Case 1: $dob(\mathsf{c_P}) < dob(\mathsf{c_Q}) = i + 1$. In this case, the witnessing axiom for $\mathsf{c_P}$ belongs to a language earlier than $\mathcal{L}_{i+1}$. So by Observation above, $\mathsf{c_Q}$ is not in it.
Case 2: $dob(\mathsf{c_P}) = dob(\mathsf{c_Q}) = i + 1$. In that case $w_i(\mathsf{P}) = \mathsf{c_P}$ and $w_i(\mathsf{Q}) = \mathsf{c_Q}$. So P and Q belong to $\mathcal{L}_i$. So $\mathsf{c_Q}$ is not in P and $\mathsf{c_P}$ is not in Q. So $\mathsf{c_Q}$ is not in $(\mathsf{Pc_P/v} \rightarrow \forall \mathsf{vP})$ and $\mathsf{c_P}$ is not in $(\mathsf{Qc_Q/v} \rightarrow \forall \mathsf{vQ})$.
(We ignore the case in which $dob(\mathsf{c_P}) = dob(\mathsf{c_Q}) = 0$, because no witness has birth date of 0.)

Finally, we are in a position to prove the Elimination Theorem.

The Elimination Theorem:
Where A is a set of sentences of $\mathcal{L}$ (the original language) and Q is a sentence of $\mathcal{L}$, if $A \cup H \vdash_{\mathcal{L}+} \mathsf{Q}$, then $A \vdash \mathsf{Q}$.

Proof: By induction on the maximum number k of members of H used (cited) in the proof of Q from $A \cup H$.

Basis: $k = 0$. Trivial.

Induction hypothesis: Suppose that the thesis holds when at most k members of H are used in the derivation of Q from $A \cup H$.

Induction step: Show that the thesis holds when $k + 1$ members of H are used in the derivation. Let U be the members of H that are used.

Case 1: At least one member of U is not a witnessing axiom. (It is a sentence of one of the following forms: $(\forall \mathsf{vP} \rightarrow \mathsf{Pn/v})$, $((\mathsf{Pn/v} \wedge \mathsf{n = m}) \rightarrow \mathsf{Pm/v})$, $\mathsf{n = n}$.) By Proposition 6, we can prove that member from A and the remainder of U. This brings the number of members used down to k; so by the induction hypothesis, the thesis holds.

Case 2: All of the members of U are witnessing axioms, i.e., of the form:

$$(\mathsf{Pc_P/v} \rightarrow \forall \mathsf{vP})$$

If for some sentence of this form in U, $\mathsf{Pc_P/v} = \mathsf{P}$ (i.e., v is not free in P), then, by Proposition 5, we can prove that member from A and the remainder of U. This brings the number of members used down to k; so by the induction hypothesis, the thesis holds.

Otherwise, each member of U contains a witness for some formula. Since U is finite, we can find a witness in a sentence in U that has a latest birth date (i.e., a birth date such that for every other witness in a sentence in U, its birth date is no later). Call this witness c_{P^*}. So there is some formula P^* such that $w(P^*) = c_{P^*}$, and U contains

$$(P^*c_{P^*}/v \rightarrow \forall v P^*).$$

Since c_{P^*} is not in $\mathcal{L}$, c_{P^*} occurs in no member of A and does not occur in Q. By the Independence Lemma, c_{P^*} occurs in no other member of U either.

Let U^* be the set containing every member of U *other than* $(P^*c_{P^*}/v \rightarrow \forall v P^*)$. By Proposition 5, there is a proof of Q from $A \cup U^*$. *That* proof may contain c_{P^*} and other names not in $\mathcal{L}$; however given that proof we can certainly find another one containing exclusively names in $\mathcal{L}$. The number of members of H that are used in this proof will be no greater than k. So by the induction hypothesis, the thesis holds.

End of proof.

All that remains is to prove the Henkin Construction Theorem.

Lemma: The Equivalence of Identicals:
If *val* is a truth value assignment for $\mathcal{L}$+ such that for all $P \in H$, $Val(P) = T$ (*val satisfies H*), then for all constants n, m, and o of $\mathcal{L}$+, then
(i) $val(n = n) = T$, and
(ii) if $val(n = m) = T$ then $val(m = n) = T$, and
(iii) if $val(n = m) = T$ and $val(m = o) = T$, then $val(n = o) = T$.
(In other words, the relation between constants of *flanking a true identity* is an *equivalence* relation.)
Proof: Exercise. (Hint: Look at the sentences that have to be in H by the definition of that set.)

Define $[n] = \{m \mid m$ is a constant of $\mathcal{L}$+ and $val(n = m) = T\}$. (Call this the *equivalence class for n relative to val*.)

Proposition 7:
If for all $P \in H$, $Val(P) = T$, then $\{\langle [n], [m]\rangle \mid val(n = m) = T\}$ is an identity relation. (I.e., if $val(n = m) = T$, then $[n] = [m]$.)
Proof: Suppose not. Then there are n and m such that $val(n = m) = T$, but $[n] \neq [m]$.

Case 1: There is a constant o such that $o \in [n]$ but $o \notin [m]$. In that case, *val*(n = o) = T, but *val*(m = o) ≠ T. By Equivalence of Identicals (ii), *val*(m = n) = T. So by Equivalence of Identicals (iii), *val*(m = o) ≠ T. Contradiction.
Case 2: There is a constant o such that $o \in [m]$, but $o \notin [n]$. Similarly.

Proposition 8:
If for all $P \in H$, *Val*(P) = T and [n] = [o], then *val*(n = o) = T.
Proof: Suppose for all $P \in H$, *Val*(P) = T and [n] = [o]. So suppose, for a reductio, that *val*(n = o) ≠ T. In that case, $o \notin [n]$. But by the Equivalence of Identicals (i), $n \in [n]$. So, contrary to assumption, [n] ≠ [o].

Proposition 9: If for all $P \in H$, *Val*(P) = T and [n] = [o] and *Val*(Pn/v) = T, then *Val*(Po/v) = T. *Proof: Exercise.* (Hints: Think about the definitions of [n] and *H*.)

The Satisfaction Lemma:
For every structure $\mathfrak{M}$, every variable assignment *g*, and every formula Q, *g*[v/Σ(n)] satisfies Q in $\mathfrak{M}$ if and only if *g* satisfies Qn/v in $\mathfrak{M}$. (*g*[v/Σ(n)] assigns to v the object that Σ assigns to n.)

Proof: By induction on the complexity of wffs.

Basis: Suppose $Q = Rt_1t_2...t_m$.
Case 1: For every *i*, $1 \leq i \leq m$, $t_i \neq v$. So Qn/v = Q.
$\langle h[v/\Sigma(n)](t_1), h[v/\Sigma(n)](t_2), \ldots, h[v/\Sigma(n)](t_m)\rangle = \langle h(t_1), h(t_2), \ldots, h(t_m)\rangle$.
Case 2: For some *i*, $1 \leq i \leq m$, $t_i = v$. So $Qn/v = Rt_1t_2...n\ldots t_m$.
$\langle h[v/\Sigma(n)](t_1), \ldots, h[v/\Sigma(n)](v), \ldots, h[v/\Sigma(n)](t_m)\rangle = \langle h(t_1), \ldots, \Sigma(n),\ldots, h(t_m)\rangle = \langle h(t_1), \ldots, h(n),\ldots, h(t_m)\rangle$.

Induction hypothesis: Suppose the thesis holds for Q and R.

Induction step:
(¬): Suppose P = ¬Q. Since, by the induction hypothesis, *g*[v/Σ(n)] satisfies Q in $\mathfrak{M}$ if and only if *g* satisfies Qn/v in $\mathfrak{M}$, *g*[v/Σ(n)] satisfies P in $\mathfrak{M}$ if and only if *g* satisfies Pn/v in $\mathfrak{M}$.
(v): Similarly.
(∀): Suppose P = ∀uQ.
Case (i): u ≠ v. By the induction hypothesis, for all $o \in D_M$, *g*[v/Σ(n)][u/*o*] satisfies Q if and only *g*[u/*o*] satisfies Qn/v. So *g*[v/Σ(n)] satisfies ∀uQ if and only if *g* satisfies ∀u[Qn/v] = [∀uQ]n/v. (This identity holds because u ≠ v.) So *g*[v/Σ(n)] satisfies P if and only if *g* satisfies Pn/v.

Case (ii): u = v. So $g[v/\Sigma(n)][u/o] = g[u/o]$. So, trivially, for all $o \in D_M$, $g[v/\Sigma(n)][u/o]$ satisfies Q if and only if $g[u/o]$ satisfies Q. So $g[v/\Sigma(n)]$ satisfies ∀uQ if and only if g satisfies ∀uQ. But also P = ∀uQ = (since v = u is bound) [∀uQ]n/v = Pn/v. So $g[v/\Sigma(n)]$ satisfies P if and only if g satisfies Pn/v.

Next, we will do an inductive proof that uses the concept of the complexity of a sentence. So first, we need to extend the definition of complexity from Lesson 2 to cover the case of quantified sentences (and swap → for ∨). Thus:

Atomic sentences have complexity 0.
If a sentence P has complexity n, then ¬P has complexity n+1.
If out of the two sentences P and Q, the complexity of the one with the greatest complexity is n, then the complexity of (P → Q) is n+1.
If Qn/v has complexity n, then ∀vQ has complexity n + 1.

The Henkin Construction Theorem:
For every truth value assignment *val*, if for all P ∈ *H*, *Val*(P) = T (i.e, *val satisfies H*), then there is a structure $\mathfrak{M}_{val}$ such that for all P of $\mathcal{L}$+, if *Val*(P) = T, then P is true in $\mathfrak{M}_{val}$.

Proof: Suppose that truth value assignment *val* that satisfies *H*. We show how to construct $\mathfrak{M}_{val}$ and then we prove something stronger, viz., for every sentence P of $\mathcal{L}$+, *Val*(P) = T *if and* only if P is true in $\mathfrak{M}_{val}$. We prove the latter by induction on the complexity of sentences.

First part: Construction of $\mathfrak{M}_{val}$.
$\mathfrak{M}_{val} = \langle D, \Sigma \rangle$.
D = {[n] | n is a constant of $\mathcal{L}$+}. ([n] is defined in terms of the given *val*.)
For every constant n of $\mathcal{L}$+, Σ(n) = [n].
For every m-place predicate R of $\mathcal{L}$+,
$\Sigma(R) = \{\langle [n_1], [n_2], \ldots, [n_m] \rangle \mid val(Rn_1n_2...n_m) = T\}$.

Nota bene: We are interpreting our language in a domain of objects that are equivalence classes of names of that same language!

We have to check to make sure that $\mathfrak{M}_{val}$ so defined really is a structure for $\mathcal{L}$+. The domain of $\mathfrak{M}_{val}$ is nonempty and, by Proposition 7, $\Sigma(=)$ is the identity relation on the domain of $\mathfrak{M}_{val}$. So yes, it is.

Second part: The induction. We prove that $Val(\mathsf{P}) = \mathsf{T}$ if and only if P is true in $\mathfrak{M}_{val}$.

Basis: Suppose P is atomic; i.e., $\mathsf{P} = \mathsf{R}\mathsf{n}_1\mathsf{n}_2...\mathsf{n}_m$.

Left-to-right: Suppose $Val(\mathsf{P}) = val(\mathsf{R}\mathsf{n}_1\mathsf{n}_2...\mathsf{n}_m) = \mathsf{T}$. By the construction of $\mathfrak{M}_{val}$, $\langle[\mathsf{n}_1], [\mathsf{n}_2], \ldots, [\mathsf{n}_m]\rangle \in \Sigma(\mathsf{R})$, and so $\langle\Sigma(\mathsf{n}_1), \Sigma(\mathsf{n}_2), \ldots, \Sigma(\mathsf{n}_m)\rangle \in \Sigma(\mathsf{R})$. So $\mathsf{R}\mathsf{n}_1\mathsf{n}_2...\mathsf{n}_m = \mathsf{P}$ is true in $\mathfrak{M}_{val}$.

Right-to-left: Suppose $\mathsf{P} = \mathsf{R}\mathsf{n}_1\mathsf{n}_2...\mathsf{n}_m$ is true in $\mathfrak{M}_{val}$. By the definition of Σ, $\langle\Sigma(\mathsf{n}_1), \Sigma(\mathsf{n}_2), \ldots, \Sigma(\mathsf{n}_m)\rangle = \langle[\mathsf{n}_1], [\mathsf{n}_2],\ldots, [\mathsf{n}_m]\rangle \in \Sigma(\mathsf{R})$. So by the definition of $\Sigma(\mathsf{R})$, there are names $\mathsf{o}_1, \mathsf{o}_2, \ldots \mathsf{o}_m$, such that $[\mathsf{o}_1] = [\mathsf{n}_1]$, $[\mathsf{o}_2] = [\mathsf{n}_2]$, …, $[\mathsf{o}_m] = [\mathsf{n}_m]$, and $val(\mathsf{R}\mathsf{o}_1\mathsf{o}_2...\mathsf{o}_m) = \mathsf{T}$. But in that case, by Proposition 9, $val(\mathsf{R}\mathsf{n}_1\mathsf{n}_2...\mathsf{n}_m) = Val(\mathsf{P}) = \mathsf{T}$.

Induction hypothesis: Suppose that the thesis holds for sentences having complexity less than or equal to k.

Induction step: Show that the thesis holds for sentences having complexity $k + 1$.

Case $\neg$: $\mathsf{P} = \neg\mathsf{Q}$. *Exercise.*

Case $\rightarrow$: $\mathsf{P} = (\mathsf{Q} \rightarrow \mathsf{R})$. $Val((\mathsf{Q} \rightarrow \mathsf{R})) = \mathsf{T}$ if and only if $Val(\mathsf{Q}) = \mathsf{F}$ or $Val(\mathsf{R}) = \mathsf{T}$, which (by the induction hypothesis) is so if and only if Q is false in $\mathfrak{M}_{val}$ or R is true in $\mathfrak{M}_{val}$, which is so if and only if $(\mathsf{Q} \rightarrow \mathsf{R})$ is true in $\mathfrak{M}_{val}$.

Case $\forall$: $\mathsf{P} = \forall\mathsf{v}\mathsf{Q}$. We need to show that $Val(\forall\mathsf{v}\mathsf{Q}) = \mathsf{T}$ if and only if $\forall\mathsf{v}\mathsf{Q}$ is true in $\mathfrak{M}_{val}$.

Left-to-right: Suppose $Val(\forall\mathsf{v}\mathsf{Q}) = \mathsf{T}$.
Since for all constants n in $\mathcal{L}$+, $(\forall\mathsf{v}\mathsf{Q} \rightarrow \mathsf{Q}\mathsf{n}/\mathsf{v}) \in H$,
for all n in $\mathcal{L}$+, $Val((\forall\mathsf{v}\mathsf{Q} \rightarrow \mathsf{Q}\mathsf{n}/\mathsf{v})) = \mathsf{T}$.
So by the definition of Val, for all n in $\mathcal{L}$+, $Val(\mathsf{Q}\mathsf{n}/\mathsf{v})) = \mathsf{T}$.
By the induction hypothesis, for all n in $\mathcal{L}$+, $\mathsf{Q}\mathsf{n}/\mathsf{v}$ is true in $\mathfrak{M}_{val}$.
So for all n in $\mathcal{L}$+, $g_{\varnothing}$ satisfies $\mathsf{Q}\mathsf{n}/\mathsf{v}$.
So, by the Satisfaction Lemma, for all n in $\mathcal{L}$+, $g_{\varnothing}[\mathsf{v}/\Sigma(\mathsf{n})]$ satisfies Q.
By the construction of $\mathfrak{M}_{val}$, for every $o \in \mathsf{D}$, there is an n in $\mathcal{L}$+ such that $\Sigma(\mathsf{n}) = o$.
So for all $o \in \mathsf{D}$, $g_{\varnothing}[\mathsf{v}/o]$ satisfies Q.

So $g_{\varnothing}$ satisfies $\forall v Q$.
So $\forall v Q$ is true in $\mathfrak{M}_{val}$.

Right-to-left: Suppose $\forall v Q$ is true in $\mathfrak{M}_{val}$.
For every $o \in D$, $g_{\varnothing}[v/o]$ satisfies Q.
By the construction of $\mathfrak{M}_{val}$, for every constant n in $\mathcal{L}+$, there is an object $o \in D$ such that $\Sigma(n) = o$.
So for all n in $\mathcal{L}+$, $g_{\varnothing}[v/\Sigma(n)]$ satisfies Q.
So, by the Satisfaction Lemma, for all n in $\mathcal{L}+$, $g_{\varnothing}$ satisfies Qn/v.
In particular, $g_{\varnothing}$ satisfies Qc_Q/v (c_Q being the witness for Q. Recall that in case v is not free in Q, Qc_Q/v is Q).
So Qc_Q/v is true in $\mathfrak{M}_{val}$.
By the induction hypothesis, $Val(Qc_Q/v) = T$.
But since $(Qc_Q/v \rightarrow \forall v Q) \in H$, $Val((Qc_Q/v \rightarrow \forall v Q)) = T$.
So, by the definition of Val, $Val(\forall v Q) = T$.

End of proof.

This completes the proof of the Completeness Theorem for first-order logic.

Exercise: Where A is a set of sentences of $\mathcal{L}$ and Q is a sentence of $\mathcal{L}$, and H is the Henkin Theory for $\mathcal{L}$ (in $\mathcal{L}+$), show that $A \vdash Q$ if and only if $A \cup H \vdash_{tt} Q$.
Hint: No inductions are called for. Assume the Soundness Theorem, as well as the Truth-functional Completeness Theorem. Use the Elimination Theorem and the stronger biconditional that we proved in order to prove the Henkin Construction Theory ($Val(P) = T$ *if* and only if P is true in $\mathfrak{M}_{val}$).

The Compactness Theorem:

Say that a set A of sentences of $\mathcal{L}$ is *first-order satisfiable* (or *first-order consistent*, or just *satisfiable*) if and only if there is a first-order structure $\mathfrak{M}$ such that every sentence in A is true in $\mathfrak{M}$ (in which case $\mathfrak{M}$ *first-order satisfies* A).

Here's the theorem: Suppose that A is a set of sentences of $\mathcal{L}$ such that for all sets of sentences B, if B is finite and $B \subseteq A$, then B is first-order satisfiable. Then A is first-order satisfiable as well.

Proof: Assume the hypothesis, and suppose, for a reductio, that A is not satisfiable.

Then $A \models \bot$.
By completeness, $A \vdash \bot$.
But proofs are finite. So there is a finite set $B \subseteq A$, such that $B \vdash \bot$.
By soundness, $B \models \bot$.
So B is not satisfiable, contrary to the supposition.
So A is satisfiable.

Alternative formulation: Suppose that A is a set of sentences of $\mathcal{L}$ and Q is a sentence of $\mathcal{L}$, and $A \models \mathsf{Q}$. Then there is a finite subset B of A such that $B \models \mathsf{Q}$.
Exercise 1: Prove the Compactness Theorem in its alternative formulation.
Exercise 2: Prove that this alternative formulation is equivalent to the first formulation. (Using the alternative formulation, prove the compactness theorem, and using the Compactness Theorem prove the alternative formulation. You do not need to use any of our "big" theorems; just use definitions. Prove each in "contrapositive form".)

This Compactness Theorem may not seem like a very exciting result, but it will be exciting to discover (as we will in Lesson 13) that satisfiability for second-order languages is *not* compact.

Some background on the concept of infinity

One kind of infinity is that of the natural numbers, 0, 1, 2, … Any set that can be put into one-one correspondence with the natural numbers is said to be *denumerable*. A set is said to be *countable* if it is either finite or denumerable. For example, the set of *even* positive integers is denumerable too, even though not all natural numbers are even:

```
0 1 2 3 …
| | | |
2 4 6 8 …
```

The set of nonnegative rational numbers is denumerable as well. Every nonnegative rational number greater than 0 appears in the following table:

	1	2	3	…
1	1/1	1/2	1/3	…
2	2/1	2/2	2/3	…
3	3/1	3/2	3/3	…
⋮	⋮	⋮	⋮	

By zig-zagging through this table, we can put the natural numbers into one-one correspondence with the nonnegative rational numbers. (Skip duplicates.)

0	1	2	3	4	5	6	…
|	|	|	|	|	|	|	
0	1/1	2/1	1/2	1/3	3/1	4/1	…

However, there *are* infinite sets that cannot be put into one-one correspondence with the natural numbers. For example, the set of all subsets of the natural numbers cannot be put into one-one correspondence with the natural numbers.

The set of a subsets of a set is called the *power set* of that set. In fact, no nonempty set can be put into one-one correspondence with its own power set. (This is known as Cantor's Theorem.)

> *Proof:* By reductio. Suppose that f is a one-one function from the members of A into its power set. Define the following set:
>
> $B = \{ n \mid n \in A \text{ and } n \notin f(n)\}$
>
> But B is a subset of A. So, by the supposition, there exists a member of A, b, such that $f(b) = B$. Question: Does b belong to B? If yes, then no. (If $b \in B$, then $b \notin f(b) = B$.) If no, then yes. (If $b \notin B$, then, since $b \in A$, b can be disqualified from membership in B only because $b \in f(b)$. But $f(b) = B$; so $b \in B$ after all.) So yes if and only if no. Contradiction! So we were mistaken in thinking that there was any such function f.

Likewise, there is no 1-1 correspondence between the natural numbers and the nonnegative *real* numbers. In fact, there is a 1-1 correspondence between the nonnegative real numbers between 0 and 1 and the power set of the set of natural numbers. (Can you prove it? Hint: Write the real numbers in base 2 and think of the 1's

and 0's as saying "yes" and "no" to each natural number. Incidentally, the set of nonnegative real numbers can be put into 1-1 correspondence with the set of *all* real numbers.)

If two sets can be put in 1-1 correspondence, then they are said to have the same *cardinality*. So the point is: There are many infinite cardinalities.

The set of sentences of $\mathcal{L}$ is denumerable. That's what we found out when we devised a method for listing the sentences of $\mathcal{L}$ one after the other (in Lesson 3). In fact, even if there are denumerably many different languages and denumerably many sentences in each of them, the set consisting of all sentences in all languages is denumerable. (*Exercise:* Think of a zig-zag procedure that would list them all.)

So now you know that there are various kinds of infinity. There's the infinity of the natural numbers (denumerable infinity), the infinity of the nonnegative real numbers (the continuum), and so on. This fact raises the following question: For any given kind of infinity, can we construct a set of sentences such that it is satisfiable only in a domain having at least that kind of infinity?

Well, we can certainly construct a set of sentences – even a very simple, finite set of sentences – that is satisfiable only in structures having denumerable domains. Consider, for instance, the following three sentences:

$\forall x \neg Rxx$
$\forall x \exists y Rxy$
$\forall x \forall y \forall z((Rxy \wedge Ryz) \rightarrow Rxz)$

To see that these three sentences are jointly satisfiable only in a domain containing at least denumerably many members, think of R as meaning "larger than".

So can we likewise construct a set of sentences that is satisfiable only in domains having least as many members as the set of nonnegative real numbers? Surprisingly, the answer is *no*. The following theorem proves it:

The (downward) Löwenheim-Skolem Theorem: If a set of sentences of $\mathcal{L}$ is first-order satisfiable, then it is first-order satisfiable in a structure with a countable domain.

Observation 1: If A is first-order satisfiable, then $A \nvdash \bot$. For if A is first-order satisfiable, then $A \not\models \bot$, which, by the soundness theorem, implies that $A \nvdash \bot$.

Observation 2: Our proof of the Henkin Construction Theorem proves something stronger: For every truth assignment *val* that tt-satisfies H, there is a first-order structure $\mathfrak{M}_{val}$ such that:
(1) for all sentences P in $\mathcal{L}$+, $Val(\mathsf{P}) = \mathsf{T}$ if and only if P is true in $\mathfrak{M}_{val}$, *and*
(2) $\mathfrak{M}_{val}$ has a countable domain (either finite or denumerable).

(2) is so, because the domain of $\mathfrak{M}_{val}$ is $\mathsf{D} = \{[\mathsf{n}] \mid \mathsf{n}$ is a constant of $\mathcal{L}$+$\}$ and there are denumerably many constants in $\mathcal{L}$+.

Proof of the (downward) Löwenheim-Skolem Theorem:
Suppose A is first-order satisfiable.
By the First Observation, $A \nvdash \bot$.
So by the Elimination Theorem (where H is the Henkin theory in $\mathcal{L}$+), $A \cup H \nvdash_{\mathcal{L}+} \bot$.
So $A \cup H \nvdash_{tt} \bot$.
By the Truth-functional Completeness Theorem, $A \cup H \not\models_{tt} \bot$.
So there is a truth value assignment *val* such that for all $\mathsf{P} \in A \cup H$, $Val(\mathsf{P}) = \mathsf{T}$.
So, by Observation 2, there is a first-order structure $\mathfrak{M}_{val}$ with a countable domain such that for all P of $\mathcal{L}$+, if $Val(\mathsf{P}) = \mathsf{T}$, then P is true in $\mathfrak{M}_{val}$.
$\mathfrak{M}_{val}$ is a first-order structure with a countable domain that first-order satisfies A.

Note 1: What Skolem actually proved, in 1919, was that if a set of sentences is first-order satisfied in a structure with a nondenumerable domain, then it is first-order satisisfied in a structure that is a *restriction* of the first to a countable domain. (For a proof, see Boolos and Jeffrey, chapter 13, or Boolos, Burgess and Jeffrey.)

Note 2: What I have here called the Löwenheim-Skolem theorem is also called the *downward* Löwenheim-Skolem theorem to distinguish it from the *upward* Löwenheim-Skolem theorem, which says that if a set A of sentences of $\mathcal{L}$ is satisfiable in any structure having an infinite domain, then for any infinite set, A is satisfiable in a structure having the same cardinality as that set. We will not prove this, but it will come up again in our discussion of second-order logic in lesson 13.

Note 3: Suppose that $\mathfrak{M}$ and $\mathfrak{N}$ are two first-order structures for a language $\mathcal{L}$. Let *map* be a 1-1 function whose domain (the set of inputs) is the domain $\mathfrak{M}$ and whose range (the set of outputs) is the domain of $\mathfrak{N}$. Then we say that $\mathfrak{M}$ and $\mathfrak{N}$ are *isomorphic* if and only if:

(i) for all constants c of $\mathcal{L}$, $\Sigma_{\mathfrak{M}}(\mathsf{c}) = o$ if and only if $\Sigma_{\mathfrak{N}}(\mathsf{c}) = map(o)$, and
(ii) for all n-ary predicates P of $\mathcal{L}$, $\langle o_1, o_2, \ldots, o_n\rangle \in \Sigma_{\mathfrak{M}}(\mathsf{P})$ if and only if $\langle map(o_1), map(o_2), \ldots, map(o_n)\rangle \in \Sigma_{\mathfrak{N}}(\mathsf{P})$.

A simple proof by induction shows that if two first order structures are isomorphic, then for any set of sentences T of $\mathcal{L}$ either both are models for T or neither is.

Suppose that T is a set of sentences such that if $\mathfrak{M}$ and $\mathfrak{N}$ are any two structures that first-order satisfy T, then $\mathfrak{M}$ and $\mathfrak{N}$ are isomorphic. This property is called *categoricity*. A set of sentences that has it is a *categorical theory*. Are there any sets of sentences that are categorical in this sense? If a set of sentences is satisfied only by structures having finite domains, then the answer is, yes. (There might be a sentence in T that tells us exactly how many objects there are.) But if a theory is satisfied only by structures having infinite domains, then the answer is, no. If a theory is satisfiable only by structures having infinite domains, then it will be satisfiable in nonisomorphic structures with infinite domains. This is an immediate consequence of the upward Löwenheim-Skolem theorem. However, in second-order languages we can write categorical sets of sentences.

Lesson 5: Preliminaries Before We Take On Gödel-incompleteness

Function symbols

We need to add some vocabulary to the language of first-order logic and augment the definition of satisfaction to allow for it. Shortly, we will be talking about the language of arithmetic, which will include vocabulary like "+" and "×", which are symbols for functions.

The syntax of function symbols:

Recall that the *terms* of a first-order language $\mathcal{L}$ include the individual constants (names) and individual variables of the language. Now we will define the set of terms of $\mathcal{L}$ as follows:

t is a *term* of $\mathcal{L}$ if and if either:
(a) t is an individual constant of $\mathcal{L}$, or
(b) t is an individual variable of $\mathcal{L}$, or
(c) f is an n-place function symbol of $\mathcal{L}$, $t_1, t_2, \ldots t_n$ are terms of $\mathcal{L}$ and $t = f(t_1, t_2, \ldots t_n)$.

For example, if "+" and "×" are 2-place function symbols of $\mathcal{L}$, then +(x, a) is a term of $\mathcal{L}$, +(b, c) is a term of $\mathcal{L}$, and ×(+(b, c), +(x, a)) is a term of $\mathcal{L}$.

For convenience, we will write +(v, u) as (v + u) and ×(v, u) as (v × u).

Wffs and sentences are defined as before, except that "terms" now includes terms of the new kind.

The semantics of function symbols:

Where **D** is a set of objects (a domain), we say that *fun* is an n-ary function on **D** if and only if *fun* is a set of n+1-tuples of members of **D** such that for all x and y, if $\langle o_1, o_2, \ldots, o_n, x\rangle \in fun$ and $\langle o_1, o_2, \ldots, o_n, y\rangle \in fun$, then $x = y$.

For example, in the domain of natural numbers the function *addition* = {⟨0, 0, 0⟩, ⟨0, 1, 1⟩, ⟨1, 0, 1⟩, ⟨0, 2, 2⟩, ⟨1, 1, 2⟩,…}.

To accommodate function symbols, we now extend the definition of an assignment, $\Sigma_{\mathfrak{M}}$, so that if f is an *n*-place function symbol of $\mathcal{L}$, then $\Sigma_{\mathfrak{M}}(\mathsf{f})$ is an *n*-ary function on $\mathbf{D}$.

We then define a term assignment *h* recursively, as follows:

Where *g* is a variable assignment in $\mathfrak{M}$, and Σ is an assignment for $\mathfrak{M}$, and t is a term of $\mathcal{L}$, $h(\mathsf{t}) = o$ if and only if either:

(i) t is an individual variable and $g(\mathsf{t}) = o$, or

(ii) t is an individual constant and $\Sigma(\mathsf{t}) = o$, or

(iii) for some terms $\mathsf{t}_1, \mathsf{t}_2, \ldots \mathsf{t}_n$ and some function symbol f, and some *n*-ary function *fun*, $\mathsf{t} = \mathsf{f}(\mathsf{t}_1, \mathsf{t}_2, \ldots \mathsf{t}_n)$, and $\Sigma(\mathsf{f}) = fun$ and $o = fun(h(\mathsf{t}_1), h(\mathsf{t}_2), \ldots, h(\mathsf{t}_n))$.

For example, if $\Sigma(\mathsf{2})$ = the number 2, and $g(\mathsf{x})$ = the number 3, and $\Sigma(\mathsf{+})$ = addition, then $h(\mathsf{+}(\mathsf{x}, \mathsf{2}))$ = the result of adding 3 and 2, i.e., 5.

The rest of the definition of satisfaction by a variable assignment in a structure (from Lesson 1) can stand without change. Likewise, the definition of truth and the definition of first-order consequence (from Lesson 1) are unchanged.

Note: Anything that can be said in a language with function symbols can be said in a language without them. Instead of a two-place function symbol +, we could have a three place predicate Add. And then, when we want to say, for example, that the sum of any number is greater than or equal to its addends, instead of saying,

$$\forall x \forall y((x + y) \geq x \wedge (x + y) \geq y),$$

we could say,

$$\forall x \forall y \forall z(\mathsf{Add}(x, y, z) \rightarrow (z \geq x \wedge z \geq y)).$$

Axiom systems

The style of doing proofs that you have learned is a "Fitch-style natural deduction system". (There are other kinds of natural deduction systems, e.g., Gentzen style. See John N. Martin's *Elements of Formal Semantics* for an approach to logic based on that.)

Another style of doing proofs is by means of an *axiom system*. These are usually easier to *state* than natural deduction systems, but much harder to *use*. But since they are easier to state, most metatheoretic work is formulated in terms of axiom systems.

Usually, an axiom system is formuated as a number of *axiom schemata* and a number of *inference rules*:

The axiom system PL (for "propositional logic"):

Three axiom schemata:
L1: $(P \rightarrow (Q \rightarrow P))$
L2: $(P \rightarrow (Q \rightarrow R)) \rightarrow ((P \rightarrow Q) \rightarrow (P \rightarrow R))$
L3: $((\neg P \rightarrow \neg Q) \rightarrow (Q \rightarrow P))$

Every wff having the form of *L1*, *L2*, or *L3* is an *axiom*. (There are infinitely many of these.)

One inference rule:
Modus Ponens (MP): Q is an *immediate consequence* of P and $(P \rightarrow Q)$.

A *proof* in PL is a finite sequence of wffs such that each member of the sequence is either an instance of L1-L3 (an axiom) or is an immediate consequence of earlier members by Modus Ponens.

The axiom system QL (Tarski 1965, Kalish and Montague 1965)

Everything in PL, plus:

Four more axiom schemata:
L4: $(\forall v(P \rightarrow Q) \rightarrow (\forall vP \rightarrow \forall vQ))$
L5: $(P \rightarrow \forall vP)$, provided v does not occur in P (vacuous quantification)
L6: $\neg \forall v \neg v = t$, where t is any term. (In other words, $\exists v\ v = t$.)
L7: $(v = t \rightarrow (P \rightarrow Q))$, where P is an atomic formula, and Q results from P by replacing any *one* occurrence of v in P with t.

One more inference rule:
Generalization: $\forall vP$ is an *immediate consequence* of P.

A *proof* in FOL is a finite sequence of wffs such that each member of the sequence is either an instance of L1-L7 (an axiom) or is an immediate consequence of earlier members by Modus Ponens or Generalization. If there is a proof having P as its last formula, we write $\vdash \mathsf{P}$. In that case we also say that P is a *theorem* of QL.

Note: In two ways this concept of proof is different from what you are accustomed to. First, there are no premises. So the conclusion of every proof is a logical truth. Second, not only sentences, but also well-formed formulas containing free variables may belong to proofs and may be the thing proved. (We can always derive a sentence from these by an application of Generalization.)

Let P be any wff (not necessarily a sentence) of a first-order language $\mathcal{L}$. (Remember: $\mathcal{L}$ contains the identity sign.) We will say that P is *first-order valid* ($\models \mathsf{P}$) if and only if for every first-order structure $\mathfrak{M}$ for $\mathcal{L}$, and every variable assignment g in $\mathfrak{M}$, P is satisfied by g in $\mathfrak{M}$.

Notice how this definition quantifies over every variable assignment rather than referring to the empty variable assignment. The reason for the change is that we now want to allow that a formula containing free variables may be first-order valid.

We can prove soundness and completeness theorems pretty much as before (although we will not bother to actually do that):
$\vdash \mathsf{P}$ if and only if $\models \mathsf{P}$.

It is often maddeningly difficult to prove even the simplest theorems.

Example 1: For all P, Q, $\vdash (\neg \mathsf{P} \rightarrow (\mathsf{P} \rightarrow \mathsf{Q}))$

1. $(\neg \mathsf{Q} \rightarrow \neg \mathsf{P}) \rightarrow (\mathsf{P} \rightarrow \mathsf{Q})$ (by L3)
2. $(((\neg \mathsf{Q} \rightarrow \neg \mathsf{P}) \rightarrow (\mathsf{P} \rightarrow \mathsf{Q})) \rightarrow (\neg \mathsf{P} \rightarrow ((\neg \mathsf{Q} \rightarrow \neg \mathsf{P}) \rightarrow (\mathsf{P} \rightarrow \mathsf{Q}))))$ (by L1)
3. $(\neg \mathsf{P} \rightarrow ((\neg \mathsf{Q} \rightarrow \neg \mathsf{P}) \rightarrow (\mathsf{P} \rightarrow \mathsf{Q})))$ (by MP from 1, 2)
4. $((\neg \mathsf{P} \rightarrow ((\neg \mathsf{Q} \rightarrow \neg \mathsf{P}) \rightarrow (\mathsf{P} \rightarrow \mathsf{Q}))) \rightarrow$
   $((\neg \mathsf{P} \rightarrow (\neg \mathsf{Q} \rightarrow \neg \mathsf{P})) \rightarrow (\neg \mathsf{P} \rightarrow (\mathsf{P} \rightarrow \mathsf{Q})))$ (by L2)
5. $((\neg \mathsf{P} \rightarrow (\neg \mathsf{Q} \rightarrow \neg \mathsf{P})) \rightarrow (\neg \mathsf{P} \rightarrow (\mathsf{P} \rightarrow \mathsf{Q})))$ (by MP from 3, 4)
6. $(\neg \mathsf{P} \rightarrow (\neg \mathsf{Q} \rightarrow \neg \mathsf{P}))$ (by F1)
7. $(\neg \mathsf{P} \rightarrow (\mathsf{P} \rightarrow \mathsf{Q}))$ (by MP from 5, 6)

For the remaining examples, I will assume, as a *theorem*, that if P is a tautology, then $\vdash \mathsf{P}$.

Example 2: If v does not occur in P, then $\vdash (\forall v(P \rightarrow Q) \rightarrow (P \rightarrow \forall vQ))$

1. $(\forall v(P \rightarrow Q) \rightarrow (\forall vP \rightarrow \forall vQ))$ (by L4)
2. $(P \rightarrow \forall vP)$ (by L5)
3. $((P \rightarrow \forall vP) \rightarrow$
   $((\forall v(P \rightarrow Q) \rightarrow (\forall vP \rightarrow \forall vQ)) \rightarrow (\forall v(P \rightarrow Q) \rightarrow (P \rightarrow \forall vQ))))$
   (by the above theorem)
4. $((\forall v(P \rightarrow Q) \rightarrow (\forall vP \rightarrow \forall vQ)) \rightarrow (\forall v(P \rightarrow Q) \rightarrow (P \rightarrow \forall vQ))))$
   (by MP from 2, 3)
5. $(\forall v(P \rightarrow Q) \rightarrow (P \rightarrow \forall vQ)))$ (by MP from 1, 4)

Example 3: $\vdash (\forall xFx \rightarrow Fa)$

1. $x = a \rightarrow (Fx \rightarrow Fa)$ (by L7)
2. $((x = a \rightarrow (Fx \rightarrow Fa)) \rightarrow (\neg Fa \rightarrow (Fx \rightarrow \neg x = a)))$ (by the theorem)
3. $(\neg Fa \rightarrow (Fx \rightarrow \neg x = a)))$ (by MP 1,2)
4. $\forall x(\neg Fa \rightarrow (Fx \rightarrow \neg x = a)))$ (by Generalization from 3)
5. $(\forall x(\neg Fa \rightarrow (Fx \rightarrow \neg x = a))) \rightarrow (\forall x\neg Fa \rightarrow \forall x(Fx \rightarrow \neg x = a)))$ (by L4)
6. $(\forall x\neg Fa \rightarrow \forall x(Fx \rightarrow \neg x = a))$ (by MP 4, 5)
7. $(\forall x(Fx \rightarrow \neg x = a) \rightarrow (\forall xFx \rightarrow \forall x \neg x = a))$ (by L4)
8. $((\forall x\neg Fa \rightarrow \forall x(Fx \rightarrow \neg x = a)) \rightarrow ((\forall x(Fx \rightarrow \neg x = a) \rightarrow$
   $(\forall xFx \rightarrow \forall x\neg x = a)) \rightarrow (\forall x\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a))))$ (by the theorem)
9. $((\forall x(Fx \rightarrow \neg x = a) \rightarrow (\forall xFx \rightarrow \forall x\neg x = a)) \rightarrow$
   $(\forall x\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a)))$ (by MP 6, 8)
10. $(\forall x\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a))$ (by MP 7, 9)
11. $\neg Fa \rightarrow \forall x\neg Fa$ (by L5)
12. $((\neg Fa \rightarrow \forall x\neg Fa) \rightarrow ((\forall x\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a)) \rightarrow$
    $(\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a))))$ (by the theorem)
13. $((\forall x\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a)) \rightarrow (\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a)))$
    (by MP 11, 12)
14. $(\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a))$ (by MP 10, 13)
15. $((\neg Fa \rightarrow (\forall xFx \rightarrow \forall x\neg x = a)) \rightarrow (\neg\forall x\neg x = a \rightarrow (\forall xFx \rightarrow Fa)))$
    (by the theorem)
16. $(\neg\forall x\neg x = a \rightarrow (\forall xFx \rightarrow Fa))$ (by MP 14, 15)
17. $\neg\forall x\neg x = a$ (by L6)
18. $(\forall xFx \rightarrow Fa)$ (by MP 16, 17)

The language of arithmetic

For purposes of coding formulas with numbers (Gödel numbering), we will want to the number of basic symbols in our language to be a prime number. So we will confine our attention to a first-order language containing just the following 13 symbols:

$$0 \;' \;(\;) \;f \;* \;v \;\neg \;\rightarrow \;\forall \;= \;\leq \;\#$$

Although, offically, the language contains just these symbols, we will use a use a number of "abbreviations" to make our meaning clearer:

1 abbreviates $0'$.
2 abbreviates $0''$.
3 abbreviates $0'''$.
⋮
x, y, etc., will abbreviate v_*, v_{**}, etc., on an ad hoc basis.
⋮
$(x + y)$ abbreviates $f_*(xy)$, which abbreviates $f_*(v_*v_{**})$.
$(x \bullet y)$ abbreviates $f_{**}(xy)$.

0 denotes the number 0 (notice the difference in font), = denotes identity, ≤ denotes the relation of being less than or equal to, + denotes addition, and • denotes multiplication.

The symbol $'$ (the "prime") denotes the successor function. So $(0' + 0''')''$ denotes the successor of the successor of the sum of the successor of 0 and the successor of the successor of the successor of 0, namely, $(1 + 3) + 2 = 6$.

I will return to the meaning of # later.

For example, the sentence

$$\forall v_*\forall v_{**}\forall v_{***}(0' \leq v_{**} \rightarrow f_{**}(v_*v_{***}) \leq f_{**}(f_*(v_*v_{**})\; v_{***}))$$

will be abbreviated as follows:

$$\forall x\forall y\forall z(1 \leq y \rightarrow (x \bullet z) \leq ((x + y) \bullet z)).$$

Since we are now using multiple vocabulary items to form a single variable and to form a single function symbol, our definition of *well-formed formula* will have to include extra clauses. We will call the language $\mathcal{L}_A$ (the "language of arithmetic").

Definition of $\mathcal{L}_A$ (*the language of arithmetic*):

f_* and f_{**} are the *two-place function symbols* of $\mathcal{L}_A$.

v followed by one or more $*$'s is a *variable* of $\mathcal{L}_A$.

t is a *term* of $\mathcal{L}_A$ if and only if either
(i) t is $\mathsf{0}$, or
(ii) v is a variable and t is v, or
(iii) t_0 is a term and t is t_0' (the term that t_0 is followed by a prime), or
(iv) t_0 and t_1 are terms, f is a two-place function symbol and t is $\mathsf{f}(t_0, t_1)$.

We will call $\mathsf{0}$ and the terms consisting of $\mathsf{0}$ followed by one or more primes *numerals.*

P is an *atomic formula* of $\mathcal{L}_a$ if and only if for some terms t_0 and t_1, P is $t_0 = t_1$, or
P is $t_0 \leq t_1$.

P is a *wff* of $\mathcal{L}_A$ if and only if either
(i) P is an atomic formula, or
(ii) for some wff Q, P is $\neg\mathsf{Q}$, or
(iii) for some wffs Q and R, P is $(\mathsf{Q} \rightarrow \mathsf{R})$, or
(iv) for some wff Q and some variable v, P is $\forall\mathsf{v}\mathsf{Q}$.

We will also use $\wedge$, $\vee$, $\leftrightarrow$ and $\exists$ as abbreviations in the usual way.

Notice that I am using the sans serif Arial font both for the language of arithmetic and for metalinguistic variables. For example, when I wrote "t is $\mathsf{0}$" in the definition of terms, "t" was a metalinguistic variable that I use to talk about the language of arithmetic, and "$\mathsf{0}$" was part of the object language, the language of arithmetic that I am talking about. You will have to determine from the context which is which.

Also, recently I have been writing "is" rather than "=", because I did not want you to confuse the "=" in the metalanguage, which I use to talk *about* $\mathcal{L}_A$ with the "$\mathsf{=}$", which is a predicate of $\mathcal{L}_A$. In the future, I will not hesitate to write "=". There is a hard-to-see difference in font (Arial for the object language, Times New Roman for the metalanguage), but you should also be able to tell from the context which identity symbol I am using.

The base 13 numbering system

We will count in a base 13 numbering system. That means that we will have single digit that means 10, a single digit that means 11 and one that means 12.

10	11	12
|	|	|
η	ε	δ
(eta)	(epsilon)	(delta)

Whereas in a base ten number system "10" denotes the number ten, in a base thirteen number system "η" denotes the number 10. In base thirteen, "10" denotes the number thirteen. "11" denotes fourteen (thirteen plus one). "1η" denotes 23 (thirteen plus ten). "δ0" denotes one hundred and fifty-six (twelve times thirteen). "δ2" denotes one hundred and fifty-eight (twelve times thirteen plus two).

We will usually be talking about numbers at a high level of abstraction; so it will not be necessary for us to learn to mentally calculate in base thirteen. It will be good enough that we understand in principle how to calculate in base thirteen

Gödel numbering

We are going to "code up" the language of arithmetic in natural numbers. An *expression* is just any sequence of symbols in the language of arithmetic — whether it forms a well-formed formula or not. With one qualification, we will assign to each expression of the language of arithmetic—indeed, to each finite sequence of expressions—a unique natural number. This assignment is called a *Gödel numbering*.

To each of the basic vocabulary items in the language of arithmetic, we will assign one of the first thirteen natural numbers (0 through 12), which we will write in base thirteen. Here is the assignment:

0	′	(	)	f	*	v	¬	→	∀	=	≤	#
|	|	|	|	|	|	|	|	|	|	|	|	|
1	0	2	3	4	5	6	7	8	9	η	ε	δ

To find the Gödel number of any complex expression, just write the numeral for the Gödel number of each of the basic vocabulary items in the same order as they occur in the expression. Thus, the Gödel number of f∗(v) (which is meaningless), in base thirteen, is 45263, i.e., writing now in base ten, $(4 \times 13^4) + (5 \times 13^3) + (2 \times 13^2) + (6 \times 13) + 3$. The Gödel number of v=v≤ (this is not a typo, just an intentionally meaningless string) in base thirteen, is 6η6ε, i.e., in base ten, $(6 \times 13^3) + (10 \times 13^2) + (6 \times 13) + 11$.

Be careful to distinguish between numbers and the numerals that denote them. In base ten, the numeral that denotes the number ten is "10". In base thirteen, the numeral that denotes the number ten is "η". In our language for arithmetic, the numeral that denotes the number ten is " 0′′′′′′′′′′ " (that's the numeral for zero followed by ten primes). A Gödel numbering is an assignment of *numbers* to expressions, not an assignment of *numerals* to expressions. But it's easy to get confused because we care a lot about the numerals that we use to denote the numbers (so much so that we switch to base 13).

Each of the numerals in the language of arithmetic itself has a Gödel number. The Gödel number of the numeral "0" is (now I'm writing in base 13) 1. The Gödel number of the numeral "0′" (notice the prime) is (now writing in base 13) 10, i.e., (now writing in base 10) 13. The Gödel number of the numeral "0′′" is (writing in base 13) 100, i.e., (writing in base 10) 13^2. In general, for any numeral consisting of "0" followed by *n* primes, the Gödel number of that numeral can be written, in base 13, with "1" followed by *n* occurrences of "0".

As I said, we also want to assign to each *sequence* of expressions a unique number. That's where the symbol "#" comes in. Instead of representing sequences of formulas in the usual way, with commas and corner brackets, thus:

⟨(Fa → Gb), Fa, Gb⟩

we will represent sequences by writing the expressions in order separated by "#"s, thus:

#(Fa→Gb)#Fa#Gb#

So now, since "#" has a Gödel number, namely, δ, we get a Gödel number for each finite sequence of expressions as well. We will call these *expressions* too. (So, looking ahead, each *proof* will have unique Gödel number, since proofs can be defined as a kind of sequence of formulas.)

Above I said that "with one exception" we would map every expression into a number. The exception is any expression of more than one symbol that begins with a prime: ′ .

The code of the prime is 0. But we cannot say that the code of ′′ is 00, because that's just 0. Likewise we cannot say that the code of ′′¬ is 007, because that's just 7, the code of ¬. So we will not assign codes to such expressions. That's not a problem since no well-formed formula begins with a prime.

Notice that because each symbol corresponds to a single digit, every *number* is the Gödel number of some expression. Leaving aside the symbols of more than one expression that begin with a prime, the mapping of expressions to natural numbers is one-one. (Systems of Gödel-numbering don't always have that property.)

Lesson 6: Sets of Numbers

Arithmetic Sets

Throughout we will assume that we are dealing with a structure in which each constant of the language of arithmetic receives its intended interpretation. So the number zero is assigned to 0, the successor function is assigned to $'$, the addition function is assigned to $+$, the multiplication function is assigned to $\bullet$, and the relation of being less than or equal to is assigned to $\leq$. The domain of the structure consists of the natural numbers, 0, 1, 2, etc. As usual, variable assignments will assign members of the domain, viz., natural numbers, to variables, and we may speak of a variable assignment as satisfying or not satisfying a formula. In speaking of satisfaction, we will not bother to mention the structure, because we assume that we are always dealing with the intended interpretation that I have just described.

We can use the language of arithmetic and the concept of satisfaction by a variable assignment to define sets of numbers. For example, we can say that the formula $3 \leq x$ (recall that this is an abbreviation of $0'' \leq v*$) defines a set of numbers, namely, those numbers n such that $g_{\varnothing}[x/n]$ satisfies $3 \leq x$. That would be, of course, the set of natural numbers greater than or equal to 3.

But since each natural number has a name in the language of arithmetic, we can get the same effect without bringing in satisfaction and variable assignments. We will use the following convention. If n is a natural number, then $\overline{n}$ is the numeral that denotes n in the $\mathcal{L}_A$, the language of arithmetic. (I could let the change in font mark the difference, but that would not work very well when we write on the blackboard.) For example, if n is 4, then $\overline{n}$ is $0''''$ (and the Gödel number of $\overline{n}$ is, in base 13, 10000). (I don't mean that if n is 4, then $\overline{n}$ and $0''''$ denote the same number; I mean that the symbol $\overline{n}$ *is* the following symbol: $0''''$.)

So we can use the formula $3 \leq x$ to define a set of numbers by using it to define the set A as follows: For all natural numbers n,

$3 \leq \overline{n}$ is true if and only if $n \in A$.

According to this definition, of course, A is the set of natural numbers greater than or equal to 3. For another example, we can define the set B as follows:

$\exists y(1 \leq y \wedge \overline{n} = (5 \bullet y))$ is true if and only if $n \in B$.

This means that B is the set of numbers that are multiples of 5 (5, 10, 15, etc.).

Let $\mathsf{F}(\mathsf{v})$ be any formula of $\mathcal{L}_A$ in which v is the sole free variable. $\mathsf{F}(\mathsf{v})$ is said to *express* a set of natural numbers A if and only if for all natural numbers n:

$\mathsf{F}(\overline{\mathsf{n}})$ is true if and only if $n \in A$.

A set of m-tuples is called an m-ary *relation*. Let $\mathsf{F}(\mathsf{v}_1, \ldots, \mathsf{v}_m)$ be a formula of $\mathcal{L}_A$ in which $\mathsf{v}_1, \ldots, \mathsf{v}_m$ are the sole free variables, where these are the first m variables in a given enumeration of the variables of $\mathcal{L}_A$, and in which all of them do occur free. Then $\mathsf{F}(\mathsf{v}_1, \ldots, \mathsf{v}_m)$ *expresses* the relation R of m-tuples if and only if:

$\mathsf{F}(\overline{\mathsf{n}}_1, \ldots, \overline{\mathsf{n}}_m)$ is true if and only if $\langle n_1, \ldots, n_m \rangle \in R$.

The reason why we require the formula that expresses a set to contain free all of the first m variables in an enumeration of the variables is that in that way we can know which "place" in an m-tuple corresponds to a given variable.

A set or relation is called *arithmetic* (pronounced with accent on third syllable) if and only if it can be expressed by a formula of $\mathcal{L}_A$.

Σ_0-formulas and sets

Within the class of arithmetic sets we can distinguish some important subsets. Toward defining a special class of arithmetic sets, we first define the concept of a Σ_0-formula. (This use of "Σ" has nothing to do with my use of it in defining first-order structures.)

An *atomic* Σ_0-formula is an atomic formula of $\mathcal{L}_A$ having one of the following four forms: $(\mathsf{c}_1 + \mathsf{c}_2) = \mathsf{c}_3$, $(\mathsf{c}_1 \bullet \mathsf{c}_2) = \mathsf{c}_3$, $\mathsf{c}_1 = \mathsf{c}_2$, or $\mathsf{c}_1 \leq \mathsf{c}_2$, where each of c_1, c_2, and c_3 is either a variable or a numeral.

Examples: $(1 + 2) = 3$, $(3 + 2) = 1$, $(1 + \mathsf{x}) = 3$, $5 = \mathsf{x}$, $5 \leq \mathsf{x}$.

We now define the set of Σ_0-formulas by means of the following four statements:

1. Every atomic Σ_0-formula is a Σ_0-formula.
2. If P and Q are Σ_0-formulas, then $\neg\mathsf{P}$ and $(\mathsf{P} \rightarrow \mathsf{Q})$ are Σ_0-formulas.
3. If P is a Σ_0-formula, v is a variable, and c is a numeral or a variable *distinct* from v, then $\forall\mathsf{v}(\mathsf{v} \leq \mathsf{c} \rightarrow \mathsf{P})$ is a Σ_0-formula.
4. Nothing else is a Σ_0-formula.

Another notational convention: We will write formulas of the form $\forall v(v \leq c \rightarrow P)$ this way: $(\forall v \leq c)P$. For example, $\forall x(x \leq 10 \rightarrow (10 \bullet x) \leq 1000))$ will be written: $(\forall x \leq 10)((10 \bullet x) \leq 1000)$. $(\exists v \leq c)P$ abbreviates $\neg(\forall v \leq c)\neg P$, which is equivalent to $\neg \forall v(v \leq c \rightarrow \neg P)$.

The expressions $(\forall v \leq c)$ and $(\exists v \leq c)$ are called *bounded quantifiers*. So we can say that Σ_0-formulas are formulas of $\mathcal{L}_A$ in which all quantification is bounded. (So we say that *quantifiers* are *bounded* (by numbers); whereas we say that *variables* are *bound* (by quantifiers).)

Notice that atomic Σ_0-formulas are defined in terms of variables and numerals, not other kinds of terms. This means that the following is *not* a Σ_0-formula: $(x + (y + z)) = w$. We can express that same (four-place) relation with a Σ_0-formula, but we have to use some devious means: $(\exists u \leq w)((y + z) = u \wedge (x + u) = w))$. (The addition of the bound on the quantifier does not change the extension, because we can be sure that none of the addends is greater than the sum.)

What is special about Σ_0-*sentences* (i.e., with no free variables) is that we can always determine whether they are true just by calculation—adding and multiplying—by operations for which there is a definite mechanical procedure. Since all quantification is bounded, we never have to do more than finitely many such calculations in order to determine whether a Σ_0-sentence is true. (Note, though, that, however little time each step takes, some finite calculations will take longer than the age of the earth to complete.)

Suppose that A is a set of natural numbers or a relation on the natural numbers expressible by some Σ_0-formula. Then A is a Σ_0 *set* or *relation*.

Let A be a Σ_0 set. Then A is *decidable* in the following sense: Given any natural number n, there is an algorithm that definitely tells us after finitely many steps either that n does belong to A or that n does not belong to A. *Proof:* Let $F(v)$ be the Σ_0-formula with one free variable, v, that expresses A. To decide whether n belongs to A, all we have to do is calculate whether the sentence $F(\bar{n})$ is true. If so, then n belongs; otherwise not. Similarly, if $F(v_1, \ldots, v_m)$ is a Σ_0-formula in which exactly $v_1, \ldots, v_m$ are free, the Σ_0-relation that it expresses is decidable (in the sense that we can decide whether any given n-tuple is a member).

Σ_1-formulas and sets

Next, we define the Σ_1 formulas, sets and relations. A *Σ_1-formula* is a formula of the form, $\exists v_{m+1} F(v_1, \ldots, v_m, v_{m+1})$, where $F(v_1, \ldots, v_m, v_{m+1})$ is a Σ_0-formula with m+1 free variables. So a Σ_1-formula begins with one unbounded existential quantifier, and all of the rest of the quantifiers in it are bounded quantifiers. A *set* or *relation* is a Σ_1 set or relation if and only if it is expressible by a Σ_1-formula.

We can define the sets and relations that are *recursively enumerable* (r.e.) to be the Σ_1 sets and relations. Here is why that is a reasonable definition. (For the moment, I confine my attention to sets, excluding relations.) Suppose A is a Σ_1 set. Then there is a Σ_0-formula $F(v, w)$ such that $\exists w F(v, w)$ expresses A. Now consider the following table:

	v	0	1	2	3	4	…
w							
0		-	✓	-	-	✓	…
1		-	✓	-	-	-	…
2		✓	-	-	-	✓	…
3		-	-	✓	-	-	…
⋮		⋮	⋮	⋮	⋮	⋮	

Let us suppose that the check marks indicate the pairs of numbers that satisfy the formula $F(v, w)$. Since $F(v, w)$ is Σ_0, we can simply calculate whether any given pair of numbers satisfies $F(v, w)$ (i.e., for any pair of numbers n and m, we can calculate whether $F(\overline{n}, \overline{m})$ is true). So by zig-zagging through this table, we can produce a list of all the numbers that satisfy $\exists w F(v, w)$ (i.e., of all of the numbers n such that $\exists w F(\overline{n}, w)$ is true). (From the table, we can tell that 0, 1, 2 and 4 do, but we can't tell yet about 3.)

Similarly, for Σ_1 *relations*. For example, let R be a set of ordered pairs; so R is a two-place relation. Let $\exists w F(u, v, w)$ be the Σ_1-formula that expresses it (so that $F(u, v, w)$ is a Σ_0-formula). Consider the following table:

	⟨u, v⟩	⟨0, 0⟩	⟨1, 0⟩	⟨0, 1⟩	⟨2, 0⟩	…
w						
0		-	✓	-	✓	…
1		-	✓	-	-	…
2		✓	-	-	✓	…
3		-	-	-	-	…
⋮		⋮	⋮	⋮	⋮	

Across the top we have a list of all *pairs* of natural numbers (which we could produce by a separate zig-zag operation). The check marks indicate *triples* that satisfy the Σ_0 fromula F(u, v, w). By zig-zagging across this table we could produce a list of all pairs of numbers that satisfy the formula ∃wF(u, v, w). (⟨0, 0⟩, ⟨1, 0⟩, and ⟨2, 0⟩ do; we can't tell yet about ⟨0, 1⟩.)

So if a set or relation is recursively enumerable in the sense defined, then we have a mechanical method, i.e., algorithm, for producing a potentially infinite list such that every member of the set is certain to eventually show up on the list. Notice, though, that a set might be recursively enumerable in this sense and still not be *decidable*. That is, we may not have a mechanical method which, given any number (or *n*-tuple of numbers), tells us whether or *not* it is a member of the set. Suppose that *A* is recursively enumerable and *n* is not in *A*. Well, we can start using our method of generating a list of the members of *A*. But there may never come a point at which we can be sure that we have gone on long enough and can conclude that *n* is not going to show up in the list. In terms of the table, there might be a column containing no check at all; but nothing tells us that if we continue zig-zagging through the table we will not eventually come upon a check in that column.

Where *A* is a set (of natural numbers), the *complement* of *A*, written $\tilde{A}$, is the set of natural numbers that do *not* belong to *A*. If *R* is a relation, i.e., a set of *n*-tuples, then the complement of *R*, written $\tilde{R}$, is the set of *n*-tuples that are *not* members of *R*.

The sets that are *recursive* (also called *recursively decidable*) can be defined as those sets or relations *R* such that both *R* and $\tilde{R}$ are recursively enumerable. Here is why that is a reasonable definition: Suppose both *R* and $\tilde{R}$ are recursively enumerable. In that case, we do have a method for deciding whether any given object *n* is a member of *R*: Alternate between listing members of *R* and listing members of $\tilde{R}$. Eventually *n* will

show up on one list or the other. If it shows up on the enumeration of R, then it is a member of R. If it shows up on the enumeration of $\tilde{R}$, then it does not belong to R.

You might find it salutary to contemplate the following fact. Suppose that $\mathsf{S}(\mathsf{x}, \mathsf{y})$ is a Σ_0-formula. Then the *relation* it expresses is recursive. But the *set* of numbers expressed by $\exists\mathsf{y}\mathsf{S}(\mathsf{x}, \mathsf{y})$ may be only recursively enumerable.

I said above that Σ_0 sets are "decidable". (I will say more about that notion of decidability in the next section.) Now I can say that Σ_0 sets are recursive (recursively decidable) in the sense just defined. Suppose a set A is Σ_0. Then there is a Σ_0-formula $\mathsf{F}(\mathsf{v})$ that expresses it. But the complement of A, $\tilde{A}$, is expressed by $\neg\mathsf{F}(\mathsf{v})$, which will be Σ_0 too. But every Σ_0-formula is Σ_1. (Where F is any Σ_0-formula and u a variable that does not occur free in F, $\exists\mathsf{u}\mathsf{F}$ is a Σ_1-formula.) So both A and $\tilde{A}$ are expressibly by Σ_1-formulas. So A is recursive.

While we are defining concepts of recursiveness, I should add (since we need to know this later): A function *fun* is a *recursive function* if and only if the relation $fun(x_1, x_2, \ldots, x_n) = x_{n+1}$ is a recursive relation.

Church's thesis

The concepts of recursively enumerability and recursiveness (recursive decidability), which I have just defined in a precise way, are the formal, i.e., precise, counterparts to the informal concepts of *effective* enumerability and decidability. (There is quite a lot of variability in terminology in the literature; so watch out.)

To say that a set is *effectively enumerable*, in the informal sense, is to say that there is some kind of *step-by-step, mechanical, brainless, stupid* procedure for generating a list of the members such that every member will eventually show up on the list. This is informal because I have not given a definition of "stupid". To say that a set is *decidable* is to say that there is some kind of step-by-step, mechanical, brainless, stupid procedure for deciding whether *or not* any given object is a member of the set.

What is known as *Church's thesis* (after the logician Alonzo Church) equates the informal with the formal notion:

Church's thesis (two parts):

(1) A set of natural numbers or relation on the natural numbers is effectively enumerable if and only if it is expressible by a Σ_1-formula (and hence recursively enumerable in the sense I defined).

(2) A set of natural numbers or relation on the natural numbers is decidable if and only both it and its complement are expressible by a Σ_1-formula (and hence recursive in the sense I defined).

The definition of recursive enumerability in terms of Σ_1 sets is just one of several possible formally precise definitions. The reason that Church's thesis seems reasonable to people is that all of the known alternative precise definiations are demonstrably equivalent to one another. Another very important one uses the concept of a Turing machine. I will not say anything more about that in this course, but anyone who wants to claim an understanding of contemporary mathematical logic needs to know about Turing machines. So I highly recommend that you read the first three or four chapters of Boolos and Jeffrey on this subject.

I have formulated Church's thesis as only a statement about sets of and relations on the natural numbers. It can be extended to other sorts of objects in so far as we have an algorithm for pairing numbers with those other sorts of objects. So, for example, it is evident that given any expression in the language of arithmetic, we can mechanically find its Gödel number and that given any number we can mechanically determine which expression it is the Gödel number of.

So we could extend Church's thesis to include the following: A set of expressions is effectively enumerable if and only if the set of Gödel numbers of the expressions in the set is expressible by a Σ_1-formula (recursively enumerable). And a set of expressions is decidable if and only if both the set of Gödel numbers of the expressions in the set and the set of Gödel numbers of expressions in the complement of the set are expressible by Σ_1-formulas (i.e., the set of Gödel numbers of expressions in the set is recursive).

Assuming Church's thesis, then, we will pretty much equate the decidability of a set of expressions with the recursiveness of a set of numbers. For example, eventually we will see that first-order logic is undecidable. That is, we will claim that there is no algorithm by which we can decide whether or not a given formula of first-order logic is valid. But what we will actually prove is that the set consisting of the Gödel numbers of valid sentences of first-order logic is not recursive (i.e., it is not the case that both that set and its complement are expressible by a Σ_1-formula).

Σ-formulas and sets

One more concept that we will need is that of a Σ-formula and a Σ set or relation (no subscript). We define the Σ-formulas by means of the following six statements:

1. Every Σ_0-formula is a Σ-formula.
2. If P is a Σ-formula and v a variable, then $\exists vP$ is a Σ-formula.
3. If P is a Σ-formula and $\overline{n}$ is a numeral and v and w are distinct variables, then all of the following are Σ-formulas: $(\forall v \leq \overline{n})P$, $(\exists v \leq \overline{n})P$, $(\forall v \leq w)P$, $(\exists v \leq w)P$.
4. If P and Q are Σ-formulas, then $(P \vee Q)$ and $(P \wedge Q)$ are Σ-formulas. (Remember that we still use $\vee$ and $\wedge$ as abbreviations.)
5. If P is a Σ_0-formula (yes, the subscript is correct) and Q is a Σ-formula, then $(P \rightarrow Q)$ is a Σ-formula.
6. Nothing else is a Σ-formula.

A Σ set or relation is a set or relation expressible by a Σ-formula.

The significance of this concept to us is as follows: Because of the connection to the concept of recursive enumerability, we will often want to know that a certain set is Σ_1. But what we will directly show is only that it is Σ. That will suffice to enable us to conclude that it is Σ_1 because of the following fact: A set or relation is Σ if and only if it is Σ_1. For the proof of this fact (the Σ sets and relations are the Σ_1 sets and relations), see Smullyan, pp. 50-53. It's not difficult, just complicated.

For purposes of understanding the proof in Smullyan, you need the following further definition: Let v_1, v_2, v_3, … be abbreviations for $v*$, $v**$, $v***$, …, respectively. A formula P is said to be *regular* if and only if for some $n \geq 1$, the free variables in P are all of v_1, v_2, … v_n. It might have been helpful for Smullyan to point out that if P is a Σ-formula that is not regular, then, where n is the largest number such that v_n occurs free in P, then we can produce a *regular* Σ-formula by writing: $(v_1 = v_1 \rightarrow (v_2 = v_2 \rightarrow \ldots \rightarrow (v_{n-1} = v_{n-1} \rightarrow P)\ldots)$. Bear that in mind when reading the last paragraph of Smullyan's proof.

Although I didn't emphasize the fact when I introduced the concept, I have been assuming that the formulas that express sets are all regular in this sense. Thus, in the case of a formula containing $v*$ and $v**$ free, we know that the formula expresses a set of pairs $\langle n, m \rangle$ such that the formula is true when we put $\overline{n}$ in place of $v*$ and $\overline{m}$ in place of $v**$. But a formula containing just $v*$ and $v***$ free, and not $v**$, does not "express" anything.

Another thing you need to know in order to read Smullyan is that where I say $\langle x_1, x_2, \ldots, x_n \rangle \in R$, Smullyan writes $R(x_1, x_2, \ldots, x_n)$.

Other sets of numbers

The Σ_1 sets are one kind of arithmetic set. Other kinds of arithmetic sets can be defined according to the complexity of the formulas of arithmetic that expresses them.

If a set of numbers is recursive, then it is very "orderly". There is an algorithm for deciding whether or not any number belongs to it. If a set is recursively enumerable, though not recursive, then it is still somewhat orderly. There may be no algorithm for deciding whether or not a given member belongs, but at least there is an algorithm for generating a list of all of the members (such that every member eventually shows up on the list).

Suppose, though, that A is not recursively enumerable but that $\mathsf{S(x,y,z)}$ is Σ_0 and the formula $\forall y \exists z \mathsf{S(x,y,z)}$ expresses A. Then A still has a kind of orderliness one step short of recursive enumerability. For each number n the set expressed by $\exists z \mathsf{S(x, \bar{n}, z)}$ is recursively enumerable. Such a set is called a Π_2 set. Suppose that B is not Π_2, but $\mathsf{T(x,y,z,w)}$ is Σ_0 and the formula $\exists y \forall z \exists w \mathsf{T(x,y,z,w)}$ expresses B. Such as set is called Σ_3. We can define a whole hierarchy of sets according to the kinds of formulas of arithmetic that express them. In the table below, assume that each of the formulas S, $\mathsf{S'}$, $\mathsf{S''}$, etc., and R, $\mathsf{R'}$, $\mathsf{R''}$, etc., is Σ_0.

	Σ_i	Π_i
$i = 0$	$\mathsf{S(x)}$	$\mathsf{R(x)}$
$i = 1$	$\exists y \mathsf{S'(x, y)}$	$\forall y \mathsf{R'(x, y)}$
$i = 2$	$\exists y \forall z \mathsf{S''(x, y, z)}$	$\forall y \exists z \mathsf{R''(x, y, z)}$
$i = 3$	$\exists y \forall z \exists w \mathsf{S'''(x, y, z, w)}$	$\forall y \exists z \forall w \mathsf{R'''(x, y, z, w)}$
⋮	⋮	⋮

The technical term for what I have here called "orderliness" is "complexity". For each i and j, if $j > i$, then Σ_j (Π_j) sets are said to be "more complex" than the Σ_i (Π_i) sets. A

whole branch of mathematical logic is devoted to the study of complexity. It is called *recursion theory*.

It is not obvious, but as a matter of fact, every arithmetic set can be expressed by a formula that belongs somewhere in this table. So every arithmetic set is one of these kinds. In the next lesson, we will see that the set of (codes of) truths of arithmetic is not arithmetic at all. In other words, the set of (codes of) truths of arithmetic is not of any of these kinds. It is as "disorderly" as could possibly be.

Lesson 7: Diagonalization

From now on, instead of speaking of the "Gödel number" of an expression in the language of arithmetic, I will usually speak of the "codes" of expressions in the language of arithmetic. (When stating major results, I will lapse back to "Gödel number".)

Consider the set consisting of all true formulas that can be written in the language of arithmetic, $\mathcal{L}_A$. Call that set $\mathcal{N}$. $\mathcal{N}$ is the set of all "truths of arithmetic". To each member of $\mathcal{N}$ there is a corresponding code (Gödel number).

In this lesson, we will prove that the set of codes of formulas in $\mathcal{N}$ is not arithmetic. In other words, there is no formula in the language of arithmetic that expresses it. This result is a significant fact in its own right. If the formulas in $\mathcal{N}$ were effectively enumerable, then, by Church's thesis, the set of codes of formulas in $\mathcal{N}$ would be recursively enumerable, which would mean that it was Σ_1. But the set of codes of formulas in $\mathcal{N}$ is not expressible by a Σ_1-formula. So $\mathcal{N}$ is not effectively enumerable, let alone decidable. There is no algorithm by which one can decide whether a sentence in the language of arithmetic is true, and there is no algorithm by which one can list, one after the other, all truths of arithmetic.

We can go further. The set of codes of truths of arithmetic is as disorderly as a set of positive integers can possible get. The set of codes of members of $\mathcal{N}$ is not expressible by any formula of the language of arithmetic at all! As I explained at the end of the last section, arithmetic sets can be placed in a hierarchy of orderliness or complexity. What we will show is that the set of codes of truths of arithmetic does not belong to *any* of these kinds. It is not arithmetic. This is called *Tarski's undefinability theorem*.

Once we have reached that conclusion, you might already be able to sniff Gödel's first incompleteness theorem just around the corner: The set of codes of formulas that are *provable* in some *theory* of arithmetic, we will find, is not so disorderly. In fact, it is Σ_1 (recursively enumerable, very orderly). So there must be some discrepancy between the set of codes of provable formulas of arithmetic and the set of codes of truths of arithmetic.

Concatenation to base 13 is Arithmetic.

Suppose we take a string of symbols, such as v0 (in this case a nonsense string), and append to it another string of symbols, such as f≤¬. The result is a longer string of

symbols: v0f≤¬. This operation is called *concatenation*. One might expect that there is a definite mathematical relation between the codes for two expressions and the code for the concatenation of those expressions. Indeed there is, and it is arithmetic. Indeed, it is Σ_0. We want to prove that fact by actually expressing that relation with a Σ_0-formula.

The code of v0 = 61 (in base 13, remember) = (6 × 13) + 1 (in base 10).
The code of f≤¬ = 4ε7 (in base 13) = $(4 \times 13^2) + (11 \times 13) + 7$ (in base 10 scientific notation).
The code of v0f≤¬ = 614ε7 (in base 13)
= $(6 \times 13^4) + (1 \times 13^3) + (4 \times 13^2) + (11 \times 13) + 7$ (in base 10)
= $((6 \times 13) + 1) \times 13^3) + (4 \times 13^2) + (11 \times 13) + 7$ (in base 10)
= $(61 \times 10^3) + 4\varepsilon 7$ (in base 13 scientific notation).
(Remember that "10" in base 13 denotes what "13" in base 10 denotes.)

Similarly, the code of v0′′ = 6100 (in base 13) = $(6 \times 13^3) + (1 \times 13^2) + (0 \times 13) + 0$ (in base 10) = $(6 \times 13^3) + 13^2$.
The code of = = η (in base 13) = 10 (in base 10).
The code of v0′′= = 6100η (in base 13)
= $(6 \times 13^4) + (1 \times 13^3) + (0 \times 13^2) + (0 \times 13) + 10$ (in base 10)
= $((6 \times 13^3) + 13^2) \times 13) + 10$ (in base 10)
= (6100 × 10) + η (in base 13).

Examining the above two examples, we detect a pattern:

Suppose that *n* is written as *m* digits in base 13. In other words, the number of digits in the base 13 numeral denoting *n* is *m*. For example, 4ε7 is written as 3 digits in base 13. We say that *m* is the *length* of *n*. So the length of 4ε7 is 3. In general, let $l(n)$ be the length of *n* written in base 13—the number of digits in the base 13 numeral denoting *n*.

In general, we can see that if *m* is the code of an expression e_1 and *n* is the code of another expression e_2, then the code for the result of concatenating e_1 with e_2 (e_1 written first) is $(m \times 10^{l(n)}) + n$ (writing in base 13). We call this relation *concatenation to base 13*. (Notice that while *concatenation* is a relation between expressions, *concatentation to base 13* is a relation between numbers).

Let us abbreviate $(m \times 10^{l(n)}) + n$ thus: $m * n$.
(Don't confuse this star with the vocabulary item of $\mathcal{L}_A$.)

Proposition 1: The relation expressed by $x * y = z$ is arithmetic; indeed it is Σ_0.

Proof:

Let $x < y$ abbreviate $x \leq y \wedge x \neq y$ (which is already an abbreviation). Let us use base 13 to abbreviate numerals of $\mathcal{L}_A$. For example, η abbreviates $0''''''''''$ (that's ten primes), and 10 abbreviates $0''''''''''''$ (that's thirteen primes).

Notice that for any positive number n, $l(n)$, the length of the base 13 numeral denoting n is simply the smallest number k such that (writing in base 13) 10^k is greater than n. For example, the length of "4ε7" is 3, and (writing in base 13) 10^3 is the smallest power of 10 greater than 4ε7. So $10^{l(y)} = x$ if and only if x is the smallest power of 10 greater than y and 1.

1. Consider the relation: x divides y (i.e., y divided by x is a whole number). That relation is expressed by: $(\exists z \leq y)((x \bullet z) = y)$. Abbreviate that as $x \text{ div } y$.

2. Consider the set of numbers x such that for some y, $x = 13^y$ (writing in base 10, the set of powers of 13). That set is expressed by the following formula. (*This only works because 13 is a prime number.*)
   $(\forall z \leq x)((z \text{ div } x \wedge z \neq 1) \rightarrow 10 \text{ div } z)$
   Abbreviate that formula as $\text{Pow}(x)$.

3. Consider the relation whose members are pairs $\langle x, y \rangle$ such that x is the smallest power of 13 greater than y and 1 (writing in base 10). That relation is arithmetic; it is expressed by:
   $(\text{Pow}(x) \wedge y < x \wedge 1 < x) \wedge (\forall z < x)((\text{Pow}(z) \wedge 1 < z) \rightarrow z \leq y))$
   Abbreviate that formula as $\text{Small}(x, y)$.

4. Consider the relation whose members are pairs $\langle x, y \rangle$ such that $10^{l(y)} = x$ (writing in base 13). By what I pointed out above, that relation is arithmetic; it is expressed by:
   $(y = 0 \wedge x = 10) \vee (y \neq 0 \wedge \text{Small}(x, y))$
   Abbreviate this as $10^{l(y)} = x$.

5. Finally, the set of triples $\langle x, y, z \rangle$ such that $(x \times 10^{l(y)}) + y = z$, i.e., $x * y = z$, is arithmetic; it is expressed by:
   $(\exists u \leq z)(\exists v \leq z)(10^{l(y)} = u \wedge (x \bullet u) = v \wedge (v + y) = z)$.
   Abbreviate this as $\text{Concat}_2(x, y, z)$.

End of proof

Let $(x * y * z)$ abbreviate $((x * y) * z)$.

Corollary: For each $n \geq 2$, the relation $(x_1 * x_2 * \ldots * x_n) = y$ is arithmetic, indeed Σ_0.

Proof: By induction.
Basis: We have proved this for $n = 2$.
Let $\mathsf{Concat}_2(x_1, x_2, y)$ be an abbeviation of the formula that expresses $(x_1 * x_2) = y$.
Induction hypothesis: The thesis holds for $n = m$.
Induction step: Show that the thesis holds for $n = m + 1$. Let $\mathsf{Concat}_m(x_1, \ldots, x_m, y)$ be an abbeviation of the formula that expresses $(x_1 * x_2 * \ldots * x_m) = y$. Show that the following formula expresses $(x_1 * x_2 * \ldots x_m * x_{m+1}) = y$.
The requisite formula is as follows:

$$(\exists z \leq y)(\mathsf{Concat}_m(x_1, \ldots, x_m, z) \wedge \mathsf{Concat}_2(z, x_{m+1}, y))$$

Note: We have proved not only that $(x_1 * x_2 * \ldots * x_n) = y$ is arithmetic, but also that it is Σ_0 (recursive), because the only quantifiers used were bounded quantifiers (but the bound may be a variable).

Some Additional Arithmetization Results

This section continues the work of the previous section in demonstrating that a number of important relations are arithmetic, indeed Σ_0. These results will not seem very interesting, but they will be used at several junctures in what follows.

6. *Begins*. We want a formula that expresses the relation that holds between numbers x and y if and only if x is the code for an expression that is the initial segment of the expression that y is the code for. For example, since $\forall v * 0$ is an initial segment of $\forall v * 0 \leq v^*$ and 9651 is the code of $\forall v * 0$ and 9651ε65 is the code of $\forall v * 0 \leq v^*$, 9651 stands in this relation to9651ε65. We call this the Begins relation. We have already defined $\mathsf{Pow}(x)$ and $\mathsf{Concat}_2(x_1, x_2, y)$ as abbreviations of Σ_0 formulas of arithmetic. Consequently, the following formula expresses the Begins relation:

$$(x = y \vee (x \neq 0 \wedge (\exists z \leq y)(\exists w \leq y)(\exists u \leq y)(\mathsf{Pow}(w) \wedge (x \bullet w) = u \wedge \mathsf{Concat}_2(u, z, y)))$$

For example, 5 begins 5007 because there is a power of 13, namely 100 (which is 13^2 written in base 13), such that $5007 = (5 \bullet 100) * 7$. (Think of 5 as x, 100 as w, 500 as u, 7 as z and 5007 as y.) Abbreviate this formula thus: xBy

7. *Ends*. Similarly, we want a formula that expresses the relation between x and y when the numeral for x is a final segment of the numeral for y. That formula is:

   $(\mathsf{x = y} \vee (\exists \mathsf{z} \leq \mathsf{y})(\mathsf{Concat}_2(\mathsf{z, x, y})))$

   Abbreviate this formula thus: xEy

8. *Part of*. This expresses the relation between x and y when the numeral for x ends some numeral that begins the numeral for y. For example, ε2 is part of 5ε294, since ε2 ends 5ε2 which begins 5ε294. This relation expressed by:

   $(\exists \mathsf{z} \leq \mathsf{y})(\mathsf{xEz} \wedge \mathsf{zBy})$

   Abbreviate this: xPy

From now on I will simply write $\mathsf{xy = z}$ as an abbreviation for $\mathsf{Concat}_2(\mathsf{x, y, z})$ (which, recall, abbreviates the formula that expresses the relation that holds between three numbers x, y and z if and only if $x * y = z$), and $\mathsf{xwy = z}$ as an abbreviation of $\mathsf{Concat}_3(\mathsf{x, w, y, z})$.

Also, I will let $\mathsf{x_1x_2\ldots x_nPy}$ abbreviate $(\exists \mathsf{z} \leq \mathsf{y})(\mathsf{Concat}_n(\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n, \mathsf{z}) \wedge \mathsf{zPy})$.

Exponentiation is Σ_1

The relation $x^y = z$ is arithmetic. Indeed, it is Σ_1. What matters for Lesson 8 will be only that exponentiation is arithmetic. But in Lesson 10, we will want to know that it is Σ_1 as well.

Toward showing that the relation $x^y = z$ is arithmetic we first need to make an observation about it that will give us the hint we need in order to write the formula that expresses it.

Proposition: $x^y = z$ if and only if there exists a set S of ordered pairs such that:
(i) $\langle y, z\rangle \in S$,
(ii) For every pair $\langle a, b\rangle \in S$, either $\langle a, b\rangle = \langle 0, 1\rangle$ or there is some pair $\langle c, d\rangle \in S$ such that $\langle a, b\rangle = \langle c + 1, d \bullet x\rangle$.

Proof:

Left-to-Right: If $x^y = z$, then we can take S to be the set $\{\langle 0,1\rangle, \langle 1, x\rangle, \langle 2, x^2\rangle, \ldots, \langle y, x^y\rangle\}$. If we do that, then (i) obviously, $\langle y, z\rangle = \langle y, x^y\rangle \in S$, and (ii) for pair $\langle a, b\rangle \in S$, either $\langle a, b\rangle = \langle 0, 1\rangle$ or there is some pair $\langle c, d\rangle \in S$ such that $\langle a, b\rangle = \langle c + 1, d \bullet x\rangle$. (For example, $\langle 2, x^2\rangle$ is not $\langle 0, 1\rangle$, but $\langle 1, x\rangle \in S$ and $\langle 2, x^2\rangle = \langle 1 + 1, x \bullet x\rangle$.

Right-to-Left: Let S be any set of ordered pairs satisfying (i) and (ii).

We prove, by induction on a that for all positive integers a, if there is an integer b such that $\langle a, b\rangle \in S$, then $x^a = b$. *Basis*: $a = 0$. If $\langle a, b\rangle \in S$ and $b = 1$, then $x^0 = 1$. If $\langle a, b\rangle = \langle 0, b\rangle \in S$, then it cannot happen that $b \neq 1$, because there is no c such that $c + 1 = 0$. *Induction hypothesis:* Suppose for arbitrary c, if there is an integer d such that $\langle c, d\rangle \in S$, then $x^c = d$. *Induction step:* Show that if there is a b such that $\langle c +1, b\rangle \in S$, then $x^{c+1} = b$. By the definition of S, if $\langle c +1, b\rangle \in S$, then there is a d such that $\langle c, d\rangle \in S$ and $\langle c +1, b\rangle = \langle c + 1, d \bullet x\rangle$. By IH, $x^c = d$. So $x^{c+1} = d \bullet x = b$.
So, given that by (1), $(y, z) \in S$, it follows that $x^y = z$.

End of proof

Now suppose we can find a Σ_0-formula $\mathsf{K(y, z, w)}$ that has the following property: For any *finite* sequence of ordered pairs of numbers $P = \langle\langle a_1, b_1\rangle, \langle a_2, b_2\rangle, \ldots, \langle a_n, b_n\rangle\rangle$, there are numbers y and z such that $y \leq w$, $z \leq w$ and $\langle y, z\rangle \in P$ if and only if $\langle y, z, w\rangle$ satisfies $\mathsf{K(y, z, w)}$.

If we can find such a formula $\mathsf{K(y, z, w)}$, then for each x and each w, we can think of the set of ordered pairs $\langle y, z\rangle$ such that $\langle y, z, w\rangle$ satisfies $\mathsf{K(y, z, w)}$ as the set S in the above proposition. Accordingly, the following Σ-formula expresses the relation $x^y = z$:

$\exists \mathsf{w}(\mathsf{K(y, z, w)} \wedge (\forall \mathsf{a} \leq \mathsf{w})(\forall \mathsf{b} \leq \mathsf{w})(\mathsf{K(a, b, w)} \rightarrow$
$((\mathsf{a = 0} \wedge \mathsf{b = 1}) \vee (\exists \mathsf{c} \leq \mathsf{a})(\exists \mathsf{d} \leq \mathsf{b})(\mathsf{K(c, d, w)} \wedge \mathsf{a = c + 1} \wedge \mathsf{b = d \bullet x}))))$

Let the formula $\mathsf{z = (x\ Exp\ y)}$ abbreviate this formula.

(Quiz: Why does $\mathsf{K(y, z, w)}$ have to be Σ_0?) So it remains to find the formula $\mathsf{K(y, z, w)}$ that does the job. Here we're going to use some tricks. The following formula defines the set of codes of terms that are expressed in base 13 notation as strings of $\mathsf{1}$'s:

$\mathsf{x \neq 0} \wedge (\forall \mathsf{y} \leq \mathsf{x})(\mathsf{yPx} \rightarrow \mathsf{1Py})$

(We stipulate that x is not 0, because as we have defined the formula P, nothing is part of 0.) Abbreviate this formula thus: Ones(x).

Where z is (a numeral for a number expressed in base 13 notation as) a string of 1's, say that the number denoted by 2z2 is a *frame*. Say that *x* is a *maximal frame* of *y* if and only if *x* is a frame, the numeral denoting *x* is a part of the numeral denoting *y*, and *x* is as long as any frame that is a part of *y*. For example, the number denoted by 21112 is a maximal frame of the number denoted by 5ε032111292δ2112. The following Σ_0-formula expresses the relation of being a maximal frame:

$$xPy \wedge (\exists z \leq y)(Ones(z) \wedge x = 2z2 \wedge \neg(\exists w \leq y)(Ones(w) \wedge 2zw2Py))$$

Abbreviate this as xMFy.

The desired formula K(y, z, w) can be written as follows:

$$(\exists u \leq w)(uMFw \wedge uuyuzuuPw \wedge \neg uPy \wedge \neg uPz)$$

This does the job, because we can think of w as denoting a sequence of pairs (by analogy, not by Gödel-numbering); we can think of uu as separating members of the sequence; we can we think of a single u as separating the members of the pairs that are members of the sequence. Since u denotes a maximal frame of the number that w denotes, and the number that u denotes is not a part of either the number that y denotes or the number that z denotes, we can be sure that all of the pairs in the sequence that we are thinking of w as denoting are included in this way. *In other words*, on analogy to the proof of Proposition above, we could prove that $\overline{z} = (\overline{x}\ Exp\ \overline{y})$ if and only if $x^y = z$.

Diagonalization is Arithmetic

For any expression, we will define an expression that we will call the *diagonal* of that expression. Likewise, we will define a relation on numbers *x* and *y* that holds just in case *x* is the code of an expression and *y* is the code of the diagonal of that expression. Finally, we will show that that relation is arithmetic.

Let E be an expression (maybe a formula, maybe not). Where v is a particular variable of the language (let's say v_*), and *n* is any number, $\forall v(v = \overline{n} \rightarrow E)$ is, of course, another expression (which is a formula if and only if E is a formula).

For any expression E, let $\#(\mathsf{E})$ be the code of that expression.

Define a two-place function *rep* as follows: Where *e* is the code of E, *rep*(*e*, *n*) is the code of $\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{n}} \rightarrow \mathsf{E})$. Call this the *representation function*.
For example, E might be the expression $2 \leq \mathsf{v}$. Then, since $0'''$ is the numeral that denotes the number 3, *rep*($\#(2 \leq \mathsf{v})$, 3) = $\#(\forall \mathsf{v}(\mathsf{v} = 0''' \rightarrow 2 \leq \mathsf{v}))$.

Define a one-place function *diag* as follows: For all numbers *x*, *diag*(*x*) = *rep*(*x*, *x*).
The numeral that denotes $\#(2 \leq \mathsf{v})$ is $\overline{\#(2 \leq \mathsf{v})}$.
So, for example, *diag*($\#(2 \leq \mathsf{v})$) = *rep*($\#(2 \leq \mathsf{v})$, $\#(2 \leq \mathsf{v})$) = $\#(\forall \mathsf{v}(\mathsf{v} = \overline{\#(2 \leq \mathsf{v})} \rightarrow 2 \leq \mathsf{v}))$.

So, in words, *diag*(*x*) is the code for the formula that results from taking the expression E for which *x* is the code, and putting the numeral for that code in place of $\overline{\mathsf{n}}$ in $\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{n}} \rightarrow \mathsf{E})$. Where $\overline{\mathsf{n}}$ is the numeral that denotes the code for E, $\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{n}} \rightarrow \mathsf{E})$ is the *diagonal* for E. In other words, the diagonal for E is $\forall \mathsf{v}(\mathsf{v} = \overline{\#\mathsf{E}} \rightarrow \mathsf{E})$, and *diag*($\#(\mathsf{E})$) is the code for that formula. Be sure to distinguish between the *diagonal* for E, which is an expression (a formula if E is a formula), and *diag*($\#(\mathsf{E})$), which is a number.

Suppose we confine our attention to the case in which E is a formula containing v as it sole free variable. Suppose also that we interpret $\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{n}} \rightarrow \mathsf{E})$ as saying, "the code for *n* satisfies E". The the diagonal of E is a sentence is a sentence that says, "my code satisfies me". If we furthermore ignore the distinction between codes and the expressions they are codes for, then the diagonal of E in effect says, "I satisfy myself."

To understand why *diag* is called the diagonal function, contemplate the following table, representing the inputs to *rep*. *rep* becomes *diag* when its inputs are restricted to those on the *diagonal* of this table:

	y	1	2	3	…
x					
1		⟨1, 1⟩	⟨1, 2⟩	⟨1, 3⟩	…
2		⟨2, 1⟩	⟨2, 2⟩	⟨2, 3⟩	…
3		⟨3, 1⟩	⟨3, 2⟩	⟨3, 3⟩	…
⋮		⋮	⋮	⋮	

Proposition 2: The function *rep* is arithmetic.

Proof: Observe that each of the components of $\forall v(v = \overline{n} \rightarrow E)$ has a definite code: The code of $\forall v(v =$ is 965265η (written in base 13), the code of $\overline{n}$ is 10^n (remember that point from Lesson 6), the code of $\rightarrow$ is 8, the code of E is *e*, and the code of) is 3. So the code of $\forall v(v = \overline{n} \rightarrow E)$ is $(965265\eta * 10^n * 8 * e * 3)$.

So the relation $rep(e, n) = y$ is expressed by the following Σ formula:

$\exists w(w = (10 \text{ Exp } n) \wedge \text{Concat}_5(965265\eta, w, 8, e, 3, y))$.

Proposition 3: The function *diag* is arithmetic.

Proof: The relation $diag(x) = y$ is expressed by the following Σ formula:

$\exists w(w = (10 \text{ Exp } x) \wedge \text{Concat}_5(965265\eta, w, 8, x, 3, y))$.

Let Diag(x, y) abbreviate the formula that expresses the relation $diag(x) = y$.

Note for later purposes: Since $z = (x \text{ Exp } y)$ is Σ, we can be sure that Diag(x, y) is Σ and therefore that the *diag* relation is Σ_1 (recursively enumerable).

Definition: For any set of numbers *A*, let *A** be the set of all *n* such that $diag(n) \in A$. (So if we find the diagonal of a code in *A*, then we put the code that it is the diagonal of in *A**. Again ignoring the distinction between codes and formulas, if there is a formula in *A* that says, "I satisfy myself!", then we put the (code for the) expression that is thus said to satisfy itself in *A**.)

Lemma 1: If *A* is arithmetic, then *A** is arithmetic.

Proof: Let A(y) be the formula that expresses *A*. Then the following formula expresses *A**:

$\exists y(\text{Diag}(x, y) \wedge A(y))$.

Definition of Gödel sentences: Where *A* is a set of numbers, G is a *Gödel sentence for A* if and only if: G is true if and only if the code for (i.e., the Gödel number of) G is in *A*.

For short: G is true $\Leftrightarrow \#(G) \in A$.

The Gödel Diagonal Lemma (lower): If A is arithmetic, then there is a Gödel sentence for A. (This is not the Gödel incompleteness theorem; we're still some distance from that. But this is itself one of the great facts of 20th century mathematical logic.)

Proof: Suppose A is arithmetic. By Lemma 1, A^* is arithmetic. Let $\mathsf{H}(\mathsf{v})$ be the formula that expresses A^*. Let h be the code of $\mathsf{H}(\mathsf{v})$, and $\overline{\mathsf{h}}$ the numeral denoting h. Then:

$\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{h}} \rightarrow \mathsf{H}(\mathsf{v}))$ is true if and only if $\mathsf{H}(\overline{\mathsf{h}})$ is true (by first-order logic).
$\mathsf{H}(\overline{\mathsf{h}})$ is true if and only if $h \in A^*$ (because $\mathsf{H}(\mathsf{v})$ expresses A^*).
$h \in A^*$ if and only if $diag(h) \in A$ (by the definition of A^*).

So: $\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{h}} \rightarrow \mathsf{H}(\mathsf{v}))$ is true if and only if $diag(h) \in A$.

But $diag(h)$ is the code for $\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{h}} \rightarrow \mathsf{H}(\mathsf{v}))$. So $\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{h}} \rightarrow \mathsf{H}(\mathsf{v}))$ is a Gödel sentence for A.

I call this the "Lower Diagonal Lemma" (a term I just made up) because it is a consequence of a more general theorem that I will introduce later on (Lesson 10), which I will call the "Upper Diagonal Lemma".

Lemma 2: If a set A is arithmetic, then so is its complement (the set of numbers *not* in A), $\tilde{A}$. *Proof:* The negation of the formula that expresses A expresses $\tilde{A}$.

The Undefinability of Arithmetic (Tarski's Undefinability Theorem) (drum roll, trumpets): The set of codes of formulas in $\mathcal{N}$ is not arithmetic.

In other words, the set of Gödel numbers of the true formulas of $\mathcal{L}_A$ is not arithmetic.

Proof:

Let T be the set of codes of true formulas of $\mathcal{L}_A$.
Suppose, for a reductio, that T is arithmetic.
Then by Lemma 2, the complement of T, $\tilde{T}$, is arithmetic too.
($\tilde{T}$ comprises the codes of expressions of $\mathcal{L}_A$ that are not formulas and the codes of formulas of $\mathcal{L}_A$ that are not true.)
By the Gödel Diagonal Lemma, there is a Gödel sentence G for $\tilde{T}$.
Since G is a Gödel sentence for $\tilde{T}$, G is true if and only if $\#(\mathsf{G}) \in \tilde{T}$.
So G is true if and only if it is not true. Contradiction!

Lesson 8: Arithmetization of Syntax and the First Incompleteness Theorem

In this lesson I will define a theory of arithmetic and demonstrate that the set of codes of sentences that are provable in that theory is Σ_1 (and therefore arithmetic). Gödel's first incompleteness theorem is just a short step beyond that.

Recall that by a *theory* I just mean a set of formulas. (Formerly I have said that a theory was a set of sentences—no free variables. But now that we understand that formulas can be true, if they are satisfied by every variable assignment, we can allow that theories include formulas with free variables.) Smullyan uses the term "system" where I say "theory". Other people, e.g., Boolos and Jeffrey, use the term "theory" to mean *set of formulas closed under logical consequence*. (So if A is a theory and $A \vdash \mathsf{Q}$, then $\mathsf{Q} \in A$.) But I will not assume that every theory is closed under logical consequence.

If *Th* is a theory in some language and Q is a formula of that language, we will say that there is a *proof* of Q in *Th* if and only if there exists a finite sequence of formulas (in the pertinent language) such that every member of the sequence is either an axiom of QL (remember that concept from Lesson 5) or a member of *Th* or follows from previous lines by Modus Ponens or Generalization.

If there is a *proof* of Q in *Th*, then we will say that Q is a *theorem* of *Th*. (This is often symbolized thus: $\vdash_{Th} \mathsf{Q}$.) The set comprising the theorems of a theory *Th* is designated thus: Con(*Th*). (It does not matter now whether we think of consequence as semantic consequence or syntactic consequence; for given soundness and completeness—which we proved, but only for a different system—these two concepts of consequence are co-extensive.)

If *Th* is an *effectively decidable* set of formulas (so that, by Church's thesis, both the set of codes of *Th* and the set of codes of the complement of *Th* are Σ_1), then *Th* is an *axiom system* and the members of *Th* together with the axioms of QL are *axioms*. (Note well: It can happen that *Th* is decidable although Con(*Th*) is not.)

Peano Arithmetic

Peano arithmetic (after Giuseppe Peano, although he did not invent it), or P.A., for short, is a theory in the language of arithmetic that consists of nine specific formulas and also all of the infinitely many formulas that fit the form of one particular axiom *scheme*. This

theory qualifies as an axiom system because there is an algorithm for deciding whether or not a formula is a member of the theory: Check whether it is one of the nine specific formulas and whether it has the form of the axiom scheme.

The nine axioms of P.A.

N1: $(x' = y' \rightarrow x = y)$
N2: $\neg 0 = x'$
N3: $(x + 0) = x$
N4: $(x + y') = (x + y)'$
N5: $(x \bullet 0) = 0$
N6: $(x \bullet y') = ((x \bullet y) + x)$
N7: $(x \leq 0 \leftrightarrow x = 0)$
N8: $(x \leq y' \leftrightarrow (x \leq y \vee x = y'))$
N9: $(x \leq y \vee y \leq x)$

Think of x as an abbreviation of v_* and think of y as an abbreviation of v_{**}. If you look at these nine formulas, you will readily recognize that they are all true. That is, they are all satisfied by every variable assignment in the intended interpretation of the language of arithmetic.

The induction scheme

Where F is any formula (in the language of arithmetic) (containing perhaps variable x free, as well as perhaps other free variables), and v is any variable that does *not* occur in F, let $F_v[y]$ (notice the square brackets) abbreviate a formula of the following form:

$$\forall v(v = y \rightarrow \forall x(x = v \rightarrow F))$$

(So the subscript on $F_v[y]$ indicates a variable that does *not* occur in F.) Then every formula of the following form is an axiom of P.A.:

N10: $(F_v[0] \rightarrow (\forall x(F \rightarrow F_v[x']) \rightarrow \forall x F))$

Dispensing with the abbreviation, we can write out N10 in full as follows (remember that v is not in F at all):

$$(\forall v(v = 0 \rightarrow \forall x(x = v \rightarrow F)) \rightarrow (\forall x(F \rightarrow \forall v(v = x' \rightarrow \forall x(x = v \rightarrow F))) \rightarrow \forall x F)).$$

You can recognize this as stating a principle of induction as follows:

$\forall v(v = 0 \rightarrow \forall x(x = v \rightarrow F))$: This is equivalent to $\forall x(x = 0 \rightarrow F)$ and says that 0 satisfies F.
$\forall x(F \rightarrow \forall v(v = x' \rightarrow \forall x(x = v \rightarrow F)))$: This says that if x satisfies F, then the successor of x satisfies F. (Notice that the second occurrence of "$\forall x$" "cuts off" the first one, so that only the occurrence of "x" in "x'" is in the scope of the first occurrence of "$\forall x$".)
$\forall xF$: This says, of course, that everything satisfies F.

Good question: Why don't we just use the following as the axiom scheme?

$(F0/x \rightarrow (\forall x(F \rightarrow Fx'/x) \rightarrow \forall xFx))$

Answer: Because the more complicated one is easier to arithmetize. That is, it is easier to show that the set of codes of such formulas is expressible by a formula in the language of arithmetic.

The Arithmetization of Proof

Our objective is to find a formula of the language of arithmetic that expresses the set of codes of formulas that are provable in P.A. We will do this by defining a series of abbreviations of formulas that will be used in building up the target formula. We began the necessary series of definitions in the previous lesson, in the definitions of $Pow(w)$ (w is a power of 13), $xy = z$ (concatenation of two expressions), $x_1x_2, \ldots x_n = z$ (concatenation of n expressions), xBy (begins), xEy (ends), xPy (part of) and $x_1x_2\ldots x_nPy$.

1. *Sequences*. Recall that the symbol # is used to represent sequences of formulas and that it has Gödel number δ (the base 13 digit for the number twelve). We want to find a formula that expresses the fact that x is the code for a sequence, not the code for a formula or other expression. Let K_{11} be the set of codes whose base 13 numerals do not contain δ. The members of K_{11} are codes of expressions that are not sequences. If $\langle n_1, n_2, \ldots, n_m\rangle$ is a sequence of numbers in K_{11}, then $\delta n_1 \delta n_2 \delta \ldots \delta n_m \delta$ (i.e., the number that is written this way in base 13, substituting the numeral for n_1 for "n_1", etc.) is a *sequence number* for the sequence $\langle n_1, n_2, \ldots, n_m\rangle$. The formula that expresses the set of sequence numbers x is:

   $(\delta Bx \wedge \delta Ex \wedge \delta \neq x \wedge \neg\delta\delta Px \wedge (\forall y \leq x)(\delta 0yPx \rightarrow \delta By))$

   Abbreviate this: $Seq(x)$

To understand the last conjunct of Seq(x), recall, from Lesson 5, that, while 0 is the Gödel number of the prime symbol, ′, we do not assign Gödel numbers to expressions of more than one symbol that begin with a prime. So a sequence can contain the expression #′# but not, for example, #′′¬v#. So the number δ0δ can be part of a sequence number, but the number δ00076δ cannot be part of a sequence number. The last conjunct in Seq(x) secures that result.

2. *Membership in a sequence*. There is a relation that holds between *x* and *y* if and only if *y* is a code for a sequence (a sequence number) and *x* is the code for a member of that sequence. The following formula expresses that relation:

   (Seq(y) ∧ δxδPy ∧ ¬δPx)

   Abbreviate this: x In y

   From now on, (∀x In y) … abbreviates (∀x ≤ y)(x In y → …). Notice that this makes (∀x In y) a bounded quantifier.

3. *Earlier in a sequence.* We want a formula that expresses the relation between *x*, *y* and *z* when *z* is the sequence number for a sequence and the expression for which *x* is the code is earlier in that sequence than the expression for which *y* is the code:

   (x In z ∧ y In z ∧ (∃w ≤ z)(wBz ∧ x in w ∧ ¬ y In w)

   Abbreviate this: $x \prec_z y$

From now on, $(\exists z,w \prec_x y)$ … abbreviates $\exists z \exists w(z \prec_x y \wedge w \prec_x y \wedge \ldots)$.

4. *Formation rules:* We want a formula that expresses the relation between *x*, *y* and *z* that holds when *x* and *y* are the codes of expressions E_x and E_y and *z* is the code for the expression $(E_x \rightarrow E_y)$. The following formula expresses that relation:

   2x8y3 = z

   (Remember that 2 is the code for (, 8 the code for →, and 3 the code for).)
   Abbreviate this as x imp y = z. Similarly, the relations between *x*, *y* and *z* when *z* is the code for $(E_x + E_y)$, $(E_x \bullet E_y)$, $E_x = E_y$, or $E_x \leq E_y$ can be expressed by formulas that we abbreviate as x pl y = z, x tim y = z, x id y = z, and x le y = z, respectively. And the relations that hold between *x* and *y* when *y* is the code for $\neg E_x$ or E_x' can be expressed by formulas that we abbreviate as neg(x) = y and s(x) = y, respectively.

From now on, I first give the abbreviation, I then state in English the relation to be expressed, and I then give the formula that expresses it.

5. St(x): *x* is the code of a string of stars (asterisks): $(\forall y \leq x)(yPx \rightarrow 5Py)$

6. Var(x): *x* is the code for a variable: $(\exists y \leq x)(St(y) \wedge 6y = x)$

7. Num(x): *x* is the code of a numeral. We already have an abbreviation for this, Pow(x), but in this context it might be helpful to use a different mnemonic. (Recall that the numerals have as their codes, 1, 10, 100, 1000, etc. (writing in base 13).)

Observe that we can represent the grammatical construction of a *term* as a sequence that starts with variables or numerals or some of both, forms terms from them, forms terms from those terms, and so on. For example, we can represent the construction of the term $((x \bullet 0') + y)$ as follows: $\langle 0', x, y, (x \bullet 0'), ((x \bullet 0') + y), ((x \bullet 0') + y)' \rangle$.

8. TermOp(x, y, z): *z* is the code for an expression that results from applying one of the basic term-forming operations to the expressions for which *x* and *y* are codes:
   $(x\ pl\ y = z \vee x\ tim\ y = z \vee s(x) = z)$

9. TermSeq(x): *x* is the sequence number for a sequence of numbers representing the formation of a term:
   $(Seq(x) \wedge (\forall y\ In\ x)(Var(y) \vee Num(y) \vee (\exists z,w \prec_x y)TermOp(z, w, y)))$
   For example, the code for the following expression will satisfy TermSeq(x):

   #x#0′#f∗(x0′)#0′′#f∗∗(0′′f∗(x0′))#

10. Term(x): *x* is the code for a term: $\exists y(TermSeq(y) \wedge x\ In\ y)$.
    Notice that this is a Σ_1-formula, but not a Σ_0-formula.

11. Atom(x): *x* is the code of an atomic formula:
    $(\exists y \leq x)(\exists z \leq x)(Term(y) \wedge Term(z) \wedge (y\ id\ z = x \vee y\ le\ z = x))$

12. Gen(x, y): *y* is the code of a universal quantification of the formula whose code is *x*: $(\exists z \leq y)(Var(z) \wedge 9zx = y)$ Notice that this doubles as a formation rule and an inference rule in our system.

At this point, observe that formulas can be built up from other formulas by any of three formula-building operations: Adding a negation sign, inserting an arrow between two

formulas and putting parentheses on the outside, or adding a universal quantifier. For example, we can represent the construction of the formula $\forall x(Fx \rightarrow Gx)$ as follows: $\langle Fx, Gx, (Fx \rightarrow Gx), \forall x(Fx \rightarrow Gx)\rangle$

13. FormOp(x, y, z): *z* is the code for an expression that results from applying one of the three formula-forming operations to the expressions for which *x* and *y* are codes:
    $x \text{ imp } y = z \vee \text{neg}(x) = z \vee \text{Gen}(x, z)$

14. FormSeq(x): *x* is the sequence number for a sequence of codes representing the formation of a formula:
    $(\text{Seq}(x) \wedge (\forall y \text{ In } x)(\text{Atom}(y) \vee (\exists z,w \prec_x y)\text{FormOp}(z, w, y)))$
    For example, the code for the following expression will satisfy FormSeq(x):

    $\#v*=0'\#v*=0''\#\neg v*=0''\#(v*=0'\rightarrow\neg v*=0'')\#$

15. Form(x): *x* is the code for a formula in the language of arithmetic:
    $\exists y(\text{FormSeq}(y) \wedge x \text{ In } y)$
    Notice that this is a Σ_1-formula, but not a Σ_0-formula.

16. Ax(x): *x* is the code of an axiom … Let's skip this for the moment and come back to it later.

17. $x \text{ imp } z = y$: We have already defined this, back at step 4, but I mention it again, because it also serves define the relation: *z* is the code for an expression derivable by Modus Ponens from the expressions for which *x* and *y* are the codes

18. Der(x, y, z): *z* is the code for an expression that is derivable by either Modus Ponens or Generalization from the expressions for which *x* and *y* are codes:
    $x \text{ imp } z = y \vee \text{Gen}(x, z)$

19. Pf(x): *x* is the sequence number of a proof in Peano arithmetic:
    $(\text{Seq}(x) \wedge (\forall y \text{ In } x)(\text{Ax}(y) \vee (\exists z,w \prec_x y)\text{Der}(z, w, y)))$

20. Prov(x): *x* the code of a formula provable in P.A.: $\exists y(\text{Pf}(y) \wedge x \text{ In } y)$.

Ta da! Prov(x) is a formula that expresses the set whose members are all and only the codes of formulas that are theorems of Peano arithmetic. (It is a Σ_1-formula, but not a Σ_0-formula.) And if you want, by cashing in all the abbreviations, you can even write it out (but there are better ways to spend your time). Except for one thing… We still have not

done item #16 above. We need to show that there is a formula that expresses the set of axioms (the usual axioms of QL and the special axioms of Peano arithmetic).

In Lesson 5, we encountered the seven axiom schemes of QL. In this lesson we have encountered the nine axioms of Peano arithmetic and the induction scheme of Peano arithmetic. This gives us seventeen axioms or axiom schemes. Suppose L1(x) is a formula that expresses the set of numbers *x* such that *x* is the code of an axiom of type L1, and L2(x) is a formula that expresses the set of numbers *x* such that *x* is the code of an axiom of type L2, …, and N1(x) is a formula that expresses the set whose sole member is the code for N1, and N2(x) is a formula that expresses the set whose sole member is the code for N2, and …, and, finally, N10(x) is a formula that expresses the set whose members are codes of formulas of the type N10 (the induction scheme). Then a formula that expresses the set of numbers *x* such that *x* is the code for an axiom is:

(L1(x) ∨ L2(x) ∨ … ∨ N1(x) ∨ N2(x) ∨ … ∨ N12(x))

This is the formula that we abbreviate as Ax(x). I won't bother to write out all nineteen of these disjuncts, but here are a few examples, including the hard cases:

L1(x) is: (∃y ≤ x)(∃w ≤ x)(∃z ≤ x) (Form(x) ∧ w imp y = z ∧ y imp z = x).

L7(z): For this one, observe that and axiom of this form can be written in the form (v = t → (X_1vX_2 → X_1tX_2). (So, in X_1vX_2, X_1 is the part of the formula that comes before v, and X_2 is the part of the formula that comes after v.) Then the formula that expresses the codes of axioms of type L7 can be expressed thus:

(∃u ≤ z)(∃t ≤ z)(∃p ≤ z)(∃q ≤ z)(∃x ≤ z)(∃y ≤ z) (∃w ≤ z) (Var(u) ∧ Term(t) ∧ Form(p) ∧ Form(q) ∧ xuy = p ∧ xty = q ∧ p imp q = w ∧ 2uηt8w3 = z).

N1(x): Let $\overline{n1}$ be the numeral that denotes the code of axiom N1. Then the formula that expresses the set containing the code of axiom N1 is simply: x = $\overline{n1}$.

N10(z): First we identify a formula Ef(v, y, f, w) that expresses the set of quadruples of codes representing variables v and y and formulas F and formulas of form ∀v(v = y → ∀x(x = v → F)), respectively, thus:

(∃x ≤ w)(Var(v) ∧ Var(y) ∧ Var(x) ∧ Form(f) ∧ ¬vPf ∧ 9v2vηy89x2xηv8f33 = w).

Then the formula that N10(z) abbreviates (satisified by the code for any instance of the induction scheme) is as follows:

$(\exists v \leq z)(\exists x \leq z)(\exists f \leq z)(\exists w_1 \leq z)(\exists w_2 \leq z)$
$(Ef(v, 0, f, w_1) \wedge Ef(v, x', f, w_2) \wedge 2w_1829x2f8w_2389xf33 = z).$

Thus, we have proved:

The Arithetization of Proof: The set of codes of provable formulas (provable in P.A.) is arithmetic.

This is so, because Prov(x) is a formula of $\mathcal{L}_A$. Indeed, we can say something stronger: The set of codes of provable formulas (provable in P.A.) is a Σ_1 set, because Prov(x) is a Σ-formula. That is evident because Prov(x) is $\exists y(Pf(y) \wedge x \text{ In } y)$, and $(Pf(y) \wedge x \text{ In } y)$ is Σ. To see that $(Pf(y) \wedge x \text{ In } y)$ is Σ, review the construction and note that we start with Σ_0 formulas and compose new formulas only in ways that conform to the definition of a Σ-formula. Since Prov(x) is a Σ-formula, and (by Smullyan's theorem) the set of codes of provable formulas is Σ_1, i.e., recursively enumerable, and the set of provable formulas is effectively enumerable.

Gödel's First Incompleteness Theorem (first formulation): There are true formulas in $\mathcal{L}_A$ that are not theorems of P.A.

First Proof (using the Undefinability of Arithmetic):

By the Undefinability of Arithmetic, the set of codes of truths of arithmetic is not arithmetic. We have seen that the set of codes of theorems of P.A. is Σ_1, and therefore, certainly, arithmetic. So, where $\mathcal{N}$ is the set of truths of arithmetic, $\mathcal{N} \neq$ Con(P.A.).

Case 1: There is a formula of arithmetic Q such that $Q \in$ Con(P.A.) and $Q \notin \mathcal{N}$. But presumably all of the theorems of P.A. are true. So this can't be right. We are left with:

Case 2: There is a formula of arithmetic Q such that $Q \notin$ Con(P.A.) and $Q \in \mathcal{N}$. Q is a truth in $\mathcal{L}_A$ that is not a theorem of P.A.

Second Proof (applying Gödel's Diagonal Lemma to the set of codes of unprovable formulas):

Let P be the set of Gödel numbers of theorems of P.A. and $\tilde{P}$ be the complement of that set, viz., the set comprising the codes of expressions that are not formulas and the codes of formulas in the language of arithmetic that are *not* theorems of P.A.

The formula $\mathsf{Prov(x)}$ defined above expresses P. So the formula $\neg\mathsf{Prov(x)}$ expresses $\tilde{P}$. So $\tilde{P}$ is arithmetic too. So by the Gödel Diagonal Lemma, there is a Gödel sentence G for $\tilde{P}$. So G is true if and only if $\#(\mathsf{G}) \in \tilde{P}$.

Case 1: G is false and $\#(\mathsf{G}) \notin \tilde{P}$. In that case, $\#(\mathsf{G}) \in P$, which means that G is a theorem of P.A. But presumably, no falsehoods are theorems of P.A. So we are left with:

Case 2: G is true and $\#(\mathsf{G}) \in \tilde{P}$, which means that G is not provable in P.A. So some truths of arithmetic are not provable in P.A.

Note: In this second proof, there is no mystery about what G says. If you wished, you could write it out. $\tilde{P}$ is arithmetic; so by Lemma 1 from Lesson 7, $\tilde{P}^*$ is arithmetic. The formula that expresses $\tilde{P}^*$ is $\exists\mathsf{y}(\mathsf{Diag(x, y)} \wedge \neg\mathsf{Prov(y)})$. Abbreviate this as $\mathsf{K(x)}$. Let k be the code of $\mathsf{K(x)}$, and $\overline{\mathsf{k}}$ the numeral denoting k. Then the Gödel G sentence for $\tilde{P}$ is $\forall\mathsf{x}(\mathsf{x} = \overline{\mathsf{k}} \rightarrow \mathsf{K(x)})$. (See the proof of the Gödel Diagonal Lemma.) Since G is true if and only if its code belongs to the set of codes of expressions that are not formulas and formulas that are not provable, popular expositions of Gödel's theorem often report that Gödel finds a sentence that says, "I am not provable". But if you think carefully about the meaning of $\mathsf{K(x)}$, it's actually not very easy to think of G as saying that, for $\mathsf{K(x)}$ does not express $\tilde{P}$ but $\tilde{P}^*$. Ignoring the difference between $\forall\mathsf{x}(\mathsf{x} = \overline{\mathsf{k}} \rightarrow \mathsf{K(x)})$ and $\mathsf{K}(\overline{\mathsf{k}})$, what G says is something more like this: "My code is in the set of codes of formulas the codes of whose diagonals are in the set of codes of expressions that either are not codes of formulas or are codes of formulas that are not provable." or, more briefly, "The diagonal of my code is not the code of a provable formula."

Note also: We have proved that G really is true (assuming that P.A. is true)! (How can we know this if we cannot derive it from P.A.?)

Why is this theorem called Gödel's *incompleteness* theorem?

Definition: Say that a theory in a language is *complete* if and only if for every *sentence* in that language either it or its negation is a theorem of the theory. (We do not require that every *formula* or its negation be true. For example, we cannot assume that either $x = 5$ or $x \neq 5$ is true. That would be like assuming that either $\forall x\ x = 5$ or $\forall x\ x \neq 5$ is true.) (Obviously, this a different sense of "complete" than we speak of in saying that our deductive calculus is "complete". We did, however, encountered this notion of completeness in Lesson 3.)

Gödel's First Incompleteness Theorem (second formulation): P.A. is incomplete.

> *Proof:* G, in the second proof above, is a sentence. As we have seen, G is true and not a theorem of P.A. But every theorem of P.A. is true; so if $\neg G$ is a theorem of P.A., then $\neg G$ is true, which means that G is false, contrary to what we have seen. So $\neg G$ is not a theorem of P.A. either. So neither G nor $\neg G$ is a theorem of P.A. So P.A. is incomplete.

We have just seen that the first formulation can be strengthened to "There are true *sentences* of arithmetic that are not theorems of P.A." and that this implies the second formulation. The second formulation implies this strengthening of the first formulation, because $\mathcal{N}$ itself is a complete theory: Since $\mathcal{N}$ is complete and P.A. is incomplete, there are *sentences* in $\mathcal{N}$ that are not theorems of P.A.

We still have not formulated Gödel's first incompleteness theorem in the manner in which nowadays it is most commonly formulated. We will be able to do that after we give the following definition:

Definition: A set of sentences *S* is *correctly axiomatizable* if and only if there is a *decidable* set *A* of *true* formulas such that all members of *S* are theorems of *A*. When *S* is correctly axiomatizable in this way, we call the decidable set *A* the *nonlogical axioms* (i.e., other than those in QL),

In light of this definition, what we have shown is the following:

Gödel's First Incompleteness Theorem (third formulation): The set of truths of arithmetic is not correctly axiomatizable.

> *Proof:* Suppose that $\mathcal{N}$ is correctly axiomatizable. Then there is a decidable set of true formulas *Ax* such that every member of $\mathcal{N}$ is a theorem of *Ax*. By Church's

thesis, the set of codes of members of *Ax* is recursively decidable. In that case, we can find a Σ_1-formula $\mathsf{Ax}(x)$ that expresses that set and prove, just as we did above in the Arithmetization of Proof for P.A., that the set of theorems of *Ax* is Σ_1 and therefore arithmetic. So by the proofs of either of the other formulations, not every member of $\mathcal{N}$ is provable in *Ax*, contrary to assumption.

Note: There is reason not to be satisfied with these proofs, namely, that we have had to assume that the theorems of P.A. are all *true*. Kurt Gödel's own original proof (published in German in 1931) did *not* assume this, in fact. All he assumed was that the theory was *ω-consistent*, which means that for each formula F, if for all n, $\mathsf{F}(\overline{n})$ is true, then $\forall v\mathsf{F}(v)$ is true. (True theories need not in general be ω-consistent, since not every object has a name; but every natural number has a name; so one might expect that a theory of *arithmetic* would be ω-consistent.) So there is a place for us to try to prove something still more general. We will do that, in fact, as a short detour (in Lesson 10) on our way to proving the undecidability of first-order logic. This more general theorem will not take for granted the truth of a theory arithmetic or even its ω-consistency, but only its *consistency*.

The Enumerability of Nonformulas

Now, while we are thinking about the recursive enumerability of formulas, having shown that the set of codes of formulas is Σ_1, I want to show also that the set of codes of strings of symbols that are *not* formulas is also recursively enumerable, i.e., Σ_1. There is a small reason to do that now, and a larger reason to have that result on board for use in proving the more general form of Gödel's First Incompleteness Theorem (in Lesson 10).

The small reason to prove now that the set of codes of nonformulas is enumerable is that we still cannot quite show that Con(P.A.) is correctly axiomatizable. The interest in the Gödel's incompleteness theorem, in the third formulation above, would seem to be somewhat diminished if not even Con(P.A.) is correctly axiomatizable. The problem is that we do not yet have a proof that the set of axioms of P.A. is decidable. Intuitively, that set is decidable (just see whether a given sentence has the form of one of the axiom schemata or not). But we would like to have an honest proof. We have seen that the set of codes of axioms is Σ_1, since $\mathsf{Ax}(x)$ is Σ. So it would suffice to show that the *complement* of the set of codes of axioms is Σ_1 too. That would show that the set of codes of axioms as recursively decidable.

Inspection of the definition of $\mathsf{Ax}(x)$ reveals that the only occurrences of unbounded quantifiers are in the formulas $\mathsf{Term}(x)$ and $\mathsf{Form}(x)$. (The first of these formulas is

∃y(TermSeq(y) ∧ x In y); the second is ∃y(FormSeq(y) ∧ x In y).) So our challenge is to find equivalent formulas (i.e., expressing the same sets of numbers) containing only bounded quantifiers. Here I follow the argument on pp. 53-54 of Smullyan.

Definition: $\pi(x) = 13^{((x \times x) + x + 1)}$

Our interest in this strange function is that it will provide the bound that we can add to our existential quantifiers.

Recall from item 4 above (concerning sequences) that K_{11} is the set of codes whose base 13 numerals do not contain δ. That is, members of K_{11} are not codes for sequences, only codes of strings of symbols in the language of arithmetic.

Theorem: Suppose $\langle a_1, \ldots, a_k \rangle$ is a sequence of numbers in K_{11}, and choose n such that $k \leq n$ and for all $i \leq k$, we have $a_i \leq n$. Let $x = \delta a_1 \delta \ldots \delta a_k \delta$. Then $x \leq \pi(n)$.

Proof: Let $y = \underbrace{\delta n \delta n \ldots \delta n}_{n} \delta$.

In other words, y is a number whose numeral in base 13 consists of the numeral δ followed by n occurrences of nδ.

It is evident that $x = \delta a_1 \delta \ldots \delta a_k \delta \leq \delta n \delta n \ldots \delta n \delta = y$. So to show that $x \leq \pi(x)$, it will suffice to show that $y \leq \pi(x)$.

For any number z, let $L(z)$ be the length of the base 13 numeral denoting z. There are n occurrences of the numeral for n in the base 13 numeral for y, and there are $n+1$ occurrences of δ in that numeral. So $L(y) = (n \times L(n)) + n + 1$. But the length of the numeral for a number is never greater than the number. So $L(n) \leq n$. $L(y) \leq (n \times n) + n + 1$. Moreover, if we take the length of a numeral for a number and raise the base to the power of that length, the result is always a larger number. (For example, writing in base ten, $L(967) = 3$, and $10^3 = 1000$.) So $y \leq 13^{L(y)}$. So $y \leq 13^{((n \times n) + n + 1)} = \pi(n)$.

End of Proof

Recall that Term(x) is defined as ∃y(TermSeq(y) ∧ x In y) (item 13 above), where (TermSeq(y) ∧ x In y) is Σ_0. We now want to show (i) that the formula (∃y ≤ π(x))(TermSeq(y) ∧ x In y) expresses this set as well, and (ii) that this set is recursive (both it and its complement are Σ_1).

(i) $(\exists y \leq \pi(x))(\mathsf{TermSeq}(y) \wedge x \mathsf{\ In\ } y)$ expresses the same set as $\mathsf{Term}(x)$:

Obviously, every member of the set expressed by $(\exists y \leq \pi(x))(\mathsf{TermSeq}(y) \wedge x \mathsf{\ In\ } y)$ is in the set expressed by $\mathsf{Term}(x)$. The challenge is to show the converse.

For a given term t, suppose s is a shortest sequence (there might be several, all equally short) such that $\langle \#(t), \#(s) \rangle$ is a member of the set of pairs expressed by $(\mathsf{TermSeq}(y) \wedge x \mathsf{\ In\ } y)$.

For some $a_1, \ldots, a_k$, the code for s is $\delta a_1 \delta \ldots \delta a_k \delta$. Since s is a shortest such sequence, a_k is the code of t.

Moreover, the code for an expression is never less than the number of term-forming operations required to construct the expression. So $k \leq a_k$. And for all $i \leq k$, we have $a_i \leq a_k$. So by the Theorem, $\delta a_1 \delta \ldots \delta a_k \delta \leq \pi(a_k)$. So $\langle \#(t), \#(s) \rangle$ is a member of the set of pairs expressed by $y \leq \pi(x)$ as well. So $\#(t)$ is a member of the set expressed by $(\exists y \leq \pi(x))(\mathsf{TermSeq}(y) \wedge x \mathsf{\ In\ } y)$.

(ii) Both $(\exists y \leq \pi(x))(\mathsf{TermSeq}(y) \wedge x \mathsf{\ In\ } y)$ and its negation express Σ_1 sets (so that the set expressed is recursive):

This formula is obviously equivalent to
$\exists z(\exists y \leq z)(z = \pi(x) \wedge \mathsf{TermSeq}(y) \wedge x \mathsf{\ In\ } y)$, which in turn is equivalent to

$\exists v \exists w \exists u \exists z(\exists y \leq z)((x \bullet x) = v \wedge v + x = w \wedge w + 1 = u \wedge z = 13^u \wedge \mathsf{TermSeq}(y) \wedge x \mathsf{\ In\ } y)$

which is Σ and which, therefore, expresses a Σ_1 set. (Here we utilize the assumption that exponentiation is Σ_1.) The negation of this formula is equivalent to:

$(\forall y \leq \pi(x))(\mathsf{TermSeq}(y) \rightarrow \neg(x \mathsf{\ In\ } y))$,

which in turn is equivalent to
$\exists z(z = \pi(x) \wedge (\forall y \leq z)(\mathsf{TermSeq}(y) \rightarrow \neg(x \mathsf{\ In\ } y)))$.

Since $\mathsf{TermSeq}(y)$ is Σ_0, this formula can similarly be seen to be Σ. Therefore it expresses a Σ_1 set.

Similarly, we can show that the complement of the set of codes of formulas is Σ_1. Let NonForm(x) be a Σ_1-formula that expresses the complement of the set of codes of formulas.

Next, for each axiom scheme and each axiom, we can write a Σ-formula that expresses the complement of the set of codes of axioms that belong to type. For example, the complement of the set of codes of axioms of type L1 (see Lesson 5), is expressible by means of the following Σ-formula:

$$(\forall y \leq x)(\forall w \leq x)(\forall z \leq x)\ ((w \text{ imp } y = z \wedge y \text{ imp } z = x) \rightarrow \text{NonForm}(x))$$

(This is Σ because the antecedent of the conditional is Σ_0.) Finally, we can construct a Σ-formula that expresses the complement of the set of codes of axioms by conjoining these formulas that express the complements of the set of codes of axioms of a given type. Since the result is Σ, we know, by Smullyan's theorem, that it is expressible by a Σ_1-formula as well.

So since the both the set of codes of axioms and the complement of that set are Σ_1, the set of codes of axioms is recursively decidable.

Lesson 9: Definability in a Theory

Recall that by a theory I just mean a set of formulas. A theory that will be of use to us (in proving the undecidability of first-order logic) is the system R (after Raphael Robinson). R consists of:

Ω_1: All sentences $(\overline{m} + \overline{n}) = \overline{k}$, where $m + n = k$.
Ω_2: All sentences $(\overline{m} \bullet \overline{n}) = \overline{k}$, where $m \times n = k$.
Ω_3: All sentences $\overline{m} \neq \overline{n}$, where m and n are distinct numbers.
Ω_4: For each n, $v_* \leq \overline{n} \leftrightarrow (v_* = 0 \vee v_* = 0' \vee \ldots \vee v_* = \overline{n})$.
Ω_5: For each n, $v_* \leq \overline{n} \vee \overline{n} \leq v_*$.

Again, the "axioms" of a theory are understood to include not only the special axioms that are actually members of the theory but also the logical axioms of QL (not to be confused with Q). To say that a sentence P is *provable* in a theory A is to say that there exists a proof in which the last line is P, where a proof is a sequence of sentences in the pertinent language and which every line is either a member of A (an *axiom*) or can be derived from earlier lines by either Modus Ponens or Generalization. In the case of R, the axioms will be any of the axioms of QL as well as any of the axioms specified by Ω_1–Ω_5.

One more bit of terminology: To say that a formula is *refutable* in a theory is to say that there is a proof of the negation of that formula in the theory. If a theory is not syntactically complete, then there will be formulas that are neither provable nor refutable in the theory.

So far, we have been proving our theorems by discovering facts about *expressibility*, viz., the expressibility of sets of numbers by formulas in the language of arithmetic. In order to introduce a different way of doing things, let me first formulate in other terms the concept of expressibility. What we have been saying is that that $F(v_1, \ldots, v_m)$ *expresses* the relation R of m-tuples if and only if:

$F(\overline{n}_1, \ldots, \overline{n}_m)$ is true if and only if $\langle n_1, \ldots, n_m \rangle \in R$.

The same concept could be expressed in different words as follows: Recall that $\mathcal{N}$ is the set of truths of arithmetic. If a sentence is a truth of arithmetic, and therefore a member of $\mathcal{N}$, then it is of course provable in $\mathcal{N}$, and if a sentence in the language of arithmetic is false, so that its negation is true and therefore provable in $\mathcal{N}$, then the sentence is refutable in $\mathcal{N}$. (Of course, by Gödel's Theorem, we know that $\mathcal{N}$ will not be correctly

axiomatizable.) So we could just as well say that $\mathsf{F}(\mathsf{v}_1, \ldots, \mathsf{v}_m)$ *expresses* the relation R of m-tuples if and only if the following two conditions hold:

(1) If $\langle n_1, \ldots, n_m\rangle \in R$, then $\mathsf{F}(\overline{\mathsf{n}}_1,\ldots,\overline{\mathsf{n}}_m)$ is provable in $\mathcal{N}$.
(2) If $\langle n_1, \ldots, n_m\rangle \notin R$, then $\mathsf{F}(\overline{\mathsf{n}}_1,\ldots,\overline{\mathsf{n}}_m)$ is refutable in $\mathcal{N}$.

Now we are going to generalize the concept of expressing a set by allowing that the pertinent provability and refutability might be some theory less than all of $\mathcal{N}$. In particular, we will be interested in what is provable and refutable in R. Also, in generalizing in this way, we will substitute the word "define" for the word "express". So we will speak of *defining* a set in a theory rather than *expressing* a set. So:

A formula $\mathsf{F}(\mathsf{v})$ is said to *define a set A in a theory Th* if and only if for all numbers n, the following two conditions hold:

(1) If $n \in A$, then $\mathsf{F}(\overline{\mathsf{n}})$ is provable in *Th*.
(2) If $n \notin A$, then $\mathsf{F}(\overline{\mathsf{n}})$ is refutable in *Th*.

A formula $\mathsf{F}(\mathsf{v}_1,\ldots, \mathsf{v}_m)$ is said to *define an m-ary relation R in a theory Th* if and only if for all numbers, $n_1, \ldots, n_m$:

(1) If $\langle n_1, \ldots, n_m\rangle \in R$, then $\mathsf{F}(\overline{\mathsf{n}}_1,\ldots,\overline{\mathsf{n}}_m)$ is provable in *Th*.
(2) If $\langle n_1, \ldots, n_m\rangle \notin R$, then $\mathsf{F}(\overline{\mathsf{n}}_1,\ldots,\overline{\mathsf{n}}_m)$ is refutable in *Th*.

In other words, if a formula defines a relation in a theory, then, returning to our earlier use of the term "express", we could say that the formula *expresses* the relation *from the point of view* of the theory.

As for functions, a formula $\mathsf{F}(\mathsf{v}_1,\ldots, \mathsf{v}_m, \mathsf{v}_{m+1})$ *strongly defines an m-ary function fun in a theory Th* if and only if for all numbers, $n_1, \ldots, n_m, n_{m+1}$:

(1) If $fun(n_1, \ldots, n_m) = n_{m+1}$, then $\mathsf{F}(\overline{\mathsf{n}}_1,\ldots, \overline{\mathsf{n}}_m, \overline{\mathsf{n}}_{m+1})$ is provable in *Th*, and
(2) if $fun(n_1, \ldots, n_m) \neq n_{m+1}$, then $\mathsf{F}(\overline{\mathsf{n}}_1,\ldots, \overline{\mathsf{n}}_m, \overline{\mathsf{n}}_{m+1})$ is refutable in *Th*, and
(3) if $fun(n_1, \ldots, n_m) = n_{m+1}$, then the sentence,
$\forall \mathsf{v}(\mathsf{F}(\overline{\mathsf{n}}_1,\ldots, \overline{\mathsf{n}}_m, \mathsf{v}) \rightarrow \mathsf{v} = \overline{\mathsf{n}}_{m+1})$
is provable in *Th*.

(So "strong definability" pertains to functions only. Without condition (3), we could not say that "according to *Th*", *fun* really is a function, with a unique output for each of its arguments.)

A set (relation, function) is *definable* in a *Th* if and only if there exists a formula in the language of the theory that defines it.

What we now want to work our way up to is the following claim: All recursive sets and relations are definable in R, and all recursive functions are strongly definable in R. (If necessary, review the definitions of recursive sets and relations, i.e., recursively definable sets and relations, and recursive functions in Lesson 6.)

(As usual, terminology varies. The term "represent" may be used where here I say "define". Here I follow Smullyan.)

Exercise (easier than it sounds): Prove that a set of numbers is arithmetic if and only if it is definable in $\mathcal{N}$ (the set of truths of $\mathcal{L}_A$).

Outline of Proof:

Definition: We say that a formula $\mathsf{F}(\mathsf{v}_1, \mathsf{v}_2)$ *enumerates* a set *A in a theory Th* if and only if for every number *n*, the following conditions hold:

(1) If $n \in A$, then there is a number *m* such that $\mathsf{F}(\overline{\mathsf{n}}, \overline{\mathsf{m}})$ is provable in *Th*.
(2) If $n \notin A$, then for *every* number *m*, $\mathsf{F}(\overline{\mathsf{n}}, \overline{\mathsf{m}})$ is refutable in *Th*.

Definition: We say that a formula $\mathsf{F}(\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_m, \mathsf{v}_{m+1})$ *enumerates* a relation *R in a theory Th* if and only if for all numbers $n_1, \ldots, n_m$, the following conditions hold:

(1) If $\langle n_1, \ldots, n_m \rangle \in R$, then there is a number *n* such that $\mathsf{F}(\overline{\mathsf{n}}_1, \ldots, \overline{\mathsf{n}}_m, \overline{\mathsf{n}})$ is provable in *Th*.
(2) If $\langle n_1, \ldots, n_m \rangle \notin R$, then for *every* number *n*, $\mathsf{F}(\overline{\mathsf{n}}_1, \ldots, \overline{\mathsf{n}}_m, \overline{\mathsf{n}})$ is refutable in *Th*.

Definition: We say that a formula $\mathsf{F}(\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_m)$ *separates* relation *A* from relation *B* in a theory *Th* if and only if:

(1) If $\langle n_1, \ldots, n_m \rangle \in A$, then $\mathsf{F}(\overline{\mathsf{n}}_1, \ldots, \overline{\mathsf{n}}_m)$ is provable in *Th*.
(2) If $\langle n_1, \ldots, n_m \rangle \in B$, then $\mathsf{F}(\overline{\mathsf{n}}_1, \ldots, \overline{\mathsf{n}}_m)$ is refutable in *Th*.

(Similarly, for separation of *sets*.)

Definition: A theory is said to be Σ_0-complete if and only if all true Σ_0-sentences are provable in it.

We will utilize the following four theorems:

R is Σ_0-complete: R is Σ_0-complete.

The Enumeration Theorem: If *Th* is Σ_0-complete, then all Σ_1 relations are enumerable in *Th*.

The Separation Theorem: If all axioms of types Ω_4 and Ω_5 (in R) are provable in *Th*, then if *A* and *B* are disjoint relations enumerable in *Th*, then *A* and *B* are separable in *Th*.

The Definability Theorem: If any two disjoint Σ_1 relations are separable in *Th*, then all recursive relations are definable in *Th*.

From these four theorems, we can immediately derive:

Theorem 1: All recursive relations are definable in R.

> *Proof:* By the Definability Theorem, it suffices to show that any two disjoint Σ_1 relations are separable in R. Let *A* and *B* be two disjoint Σ_1 relations. Since R is Σ_0-complete, the Enumeration Theorem tells us that *A* and *B* are enumerable in R. So by the Separation Theorem, they are separable in R.

Note: Throughout the following proofs, I will take for granted facts about provability that depend only on the logical axioms (from Lesson 5).

R iz Σ_0-complete (proof):
(For a different presentation, see Smullyan, pp. 66-70.)

Proposition 1: If w is any variable or numeral, $(w \leq \overline{n} \leftrightarrow (w = \overline{0} \vee w = \overline{0}' \vee \ldots \vee w = \overline{n}))$ is provable in R—by Ω_4, Generalization, and Universal Elimination (a derived rule; see example 3, Lesson 5).

Proposition 2: All true atomic Σ_0-sentences are provable in R.

Case (i): Sentences of the form $\overline{m} = \overline{m}$ are provable in R because they are theorems of QL.

Case(ii): True sentences of the form $\overline{m} \leq \overline{n}$ are provable in R: Suppose that $m \leq n$. Since $\overline{m} = \overline{m}$ is provable, so is $(\overline{m} = \overline{0} \vee \overline{m} = \overline{0'} \vee \ldots \vee \overline{m} = \overline{m} \vee \ldots \vee \overline{m} = \overline{n})$. So by Proposition 1, $\overline{m} \leq \overline{n}$ is provable in R.

Case (iii): By Ω_1 and Ω_2, true sentences of the form $(\overline{m}+\overline{n}) = \overline{k}$ and $(\overline{m}\bullet\overline{n}) = \overline{k}$ are provable in R.

Proposition 3: All false atomic Σ_0-sentences are refutable in R.

Case (i): By Ω_3, false sentences of the form $\overline{m} = \overline{n}$ are refutable in R.

Case (ii): Suppose $\overline{m} \leq \overline{n}$ is false. So $\overline{m} = 0$, $\overline{m} = 0'$, …, $\overline{m} = \overline{n}$ are all false. So by case (i), they are all refutable. So $(\overline{m} = 0 \vee \overline{m} = 0' \vee \ldots \vee \overline{m} = \overline{n})$ is refutable. So by Proposition 1, $\overline{m} \leq \overline{n}$ is refutable.

Case (iii): Suppose $(\overline{m}+\overline{n}) = \overline{k}$ is false. Then for some number $p \neq k$, $m + n = p$. By Proposition 2, $(\overline{m}+\overline{n}) = \overline{p}$ is provable in R, and, by Case (i), $\overline{p} \neq \overline{k}$ is provable in R. So, by first-order logic, $(\overline{m}+\overline{n}) \neq \overline{k}$ is provable and $(\overline{m}+\overline{n}) = \overline{k}$ is refutable in R.

Case (iv): Suppose $(\overline{m}\bullet\overline{n}) = \overline{k}$ is false. Similar to Case (iii) …

Proposition 4: Suppose $F(w)$ is a Σ_0-formula having only w free. Suppose that $F(0)$, $F(0')$, …, $F(\overline{n})$, are all provable in R. Then $(\forall w \leq \overline{n})F(w)$ is provable in R.

Proof: Assume the hypothesis. Then each of $(w = 0 \rightarrow F(w))$, $(w = 0' \rightarrow F(w))$, …, $(w = \overline{n} \rightarrow F(w))$ is provable in R. So $((w = 0 \vee w = 0' \vee \ldots \vee w = \overline{n}) \rightarrow F(w))$ is provable in R. But by, Proposition 1, $(w \leq \overline{n} \rightarrow (w = \overline{0} \vee w = \overline{0'} \vee \ldots \vee w = \overline{n}))$ is provable in R. So $(w \leq \overline{n} \rightarrow F(w))$ is provable in R. So by Generalization, $\forall w(w \leq \overline{n} \rightarrow F(w))$ is provable in R. But this is $(\forall w \leq \overline{n})F(w)$.

Theorem: R is Σ_0-complete.

We prove something stronger: Every true Σ_0-sentence is provable in R and every false Σ_0-sentence is refutable in R

Proof: By induction on the complexity of sentences.

Basis: True atomic Σ_0-sentences are provable in R.A, by Proposition 2. False atomic Σ_0-sentences are refutable in R, by Proposition 3.

Induction Hypothesis: Suppose that all Σ_0-sentences having complexity less than or equal to *k* are provable if true and refutable if false.

Induction Step:

Case $\neg$: If $\neg P$ is true, then P is false, in which case, by IH, P is refutable, i.e., $\neg P$ is provable. If $\neg P$ is false, then P is true, in which case P is provable, and $\neg\neg P$ is provable.

Case $\rightarrow$: Exercise.

Case $(\forall v \leq \overline{n})$ (bounded quantifiers): Recall, from the definition of Σ_0-sentences in Lesson 6 that the only remaining case is that of sentences of the form $(\forall v \leq \overline{n})F(v)$.

Case 1: $(\forall v \leq \overline{n})F(v)$ is true. Then each of $F(0)$, $F(0')$, …, $F(\overline{n})$ is true. So by the induction hypothesis, each of them is provable in R. So by Proposition 4, $(\forall v \leq \overline{n})F(v)$ is provable in R.

Case 2: $(\forall v \leq \overline{n})F(v)$ is false. Then for at least one $m \leq n$, $F(\overline{m})$ is false and, by IH, refutable in R. So by Proposition 2, case 2, both $\overline{m} \leq \overline{n}$ and $\neg F(\overline{m})$ are provable in R. So $(\forall v \leq \overline{n})F(v)$ is refutable in R.

End of proof.

The Enumeration Theorem

We will now prove that if *Th* is Σ_0-complete, then all Σ_1 relations are enumerable in *Th*.

Proof: Suppose that *Th* is Σ_0-complete. Let *R* be any Σ_1 relation, and let $\mathsf{R}(x_1, x_2, \ldots, x_m)$ be the Σ_1-formula that expresses it. Then (by the definition of Σ_1-formulas), there is a Σ_0-formula $\mathsf{S}(x_1, x_2, \ldots, x_m, y)$ such that:

$\mathsf{R}(x_1, x_2, \ldots, x_m)$ is true if and only if $\exists y \mathsf{S}(x_1, x_2, \ldots, x_m, y)$ is true.

We show that $\mathsf{S}(x_1, x_2, \ldots, x_m, y)$ enumerates the relation *R* in *Th*.

1. Suppose that $\langle n_1, \ldots, n_m \rangle \in R$. Then $\mathsf{R}(\overline{n}_1, \ldots, \overline{n}_m)$ is true, and for some number *n*, $\mathsf{S}(\overline{n}_1, \ldots, \overline{n}_m, \overline{n})$ is true. But this is a Σ_0-sentence; so it is provable in *Th*.

2. Suppose $\langle n_1, \ldots, n_m \rangle \notin R$. Then $\mathsf{R}(\overline{n}_1, \ldots, \overline{n}_m)$ is false, and for every *n*, $\mathsf{S}(\overline{n}_1, \ldots, \overline{n}_m, \overline{n})$ is false and $\neg \mathsf{S}(\overline{n}_1, \ldots, \overline{n}_m, \overline{n})$ is true. But the latter is a Σ_0-sentence. So for every *n*, $\mathsf{S}(\overline{n}_1, \ldots, \overline{n}_m, \overline{n})$ is refutable.

End of proof.

The Separation Theorem

For simplicity, consider just the case of *sets*, as opposed to *relations*. Suppose that all axioms of types Ω_4 and Ω_5 are provable in *Th*. Suppose also that *A* and *B* are disjoint sets enumerable in *Th*. We prove that *A* and *B* are separable in *Th*, i.e., that there is a formula $\mathsf{F}(x)$ such that:

(1) If $n \in A$, then $\mathsf{F}(\overline{n})$ is provable in *Th*.
(2) If $n \in B$, then $\mathsf{F}(\overline{n})$ is refutable in *Th*.

Let $\mathsf{A}(x, y)$ be the formula that enumerates *A*, and let $\mathsf{B}(x, y)$ be the formula that enumerates *B*. We prove that $\mathsf{F}(x)$ is $\forall y(\mathsf{B}(x, y) \rightarrow (\exists z \leq y)\mathsf{A}(x, z))$, i.e., that this latter formula separates *A* and *B*.

1. Suppose that $n \in A$. We need to show that $\forall y(\mathsf{B}(\overline{n}, y) \rightarrow (\exists z \leq y)\mathsf{A}(\overline{n}, z))$ is provable in *Th*.

   Since $\mathsf{A}(x, y)$ enumerates *A*, there is some *k*, such that $\mathsf{A}(\overline{n}, \overline{k})$ is provable in *Th*.

   Since *A* and *B* are disjoint, $n \notin B$. Since $\mathsf{B}(x, y)$ enumerates *B*, for every $m \leq k$ (indeed for every *m*), $\mathsf{B}(\overline{n}, \overline{m})$ is refutable and $\neg \mathsf{B}(\overline{n}, \overline{m})$ is provable.

So each of $(y = 0 \rightarrow \neg B(\overline{n}, y))$, $(y = 0' \rightarrow \neg B(\overline{n}, y))$, …, $(y = \overline{k} \rightarrow \neg B(\overline{n}, y))$ is provable in *Th*.

So $((y = 0 \vee y = 0' \vee \ldots \vee y = \overline{k}) \rightarrow \neg B(\overline{n}, y))$ is provable in *Th*.

By Proposition 1 (which uses Ω_4), $(y \leq \overline{k} \rightarrow (y = \overline{0} \vee y = \overline{0'} \vee \ldots \vee y = \overline{k}))$ is provable in *Th*.

So $(y \leq \overline{k} \rightarrow \neg B(\overline{n}, y))$ is provable in *Th*.

So $(B(\overline{n}, y) \rightarrow \neg y \leq \overline{k})$ is provable in *Th*.

So by Ω_5, $(B(\overline{n}, y) \rightarrow \overline{k} \leq y)$ is provable in *Th*.

Since $A(\overline{n}, \overline{k})$ is provable too, $(B(\overline{n}, y) \rightarrow (\overline{k} \leq y \wedge A(\overline{n}, \overline{k})))$ is provable in *Th*.

So $\forall y(B(\overline{n}, y) \rightarrow (\exists z \leq y)A(\overline{n}, z))$ is provable in *Th*.

2. Suppose $n \in B$. We need to show that $\forall y(B(\overline{n}, y) \rightarrow (\exists z \leq y)A(\overline{n}, z))$ is refutable in *Th*.

   Since $B(x, y)$ enumerates B, there is some k, such that $B(\overline{n}, \overline{k})$ is provable in *Th*.

   Since A and B are disjoint, $n \notin A$. Since $A(x, y)$ enumerates A, for every $m \leq k$ (indeed for every m), $A(\overline{n}, \overline{m})$ is refutable and $\neg A(\overline{n}, \overline{m})$ is provable.

   Reasoning as above (using Ω_4), $(z \leq \overline{k} \rightarrow \neg A(\overline{n}, z))$ is provable in *Th*. So by Generalization, $(\forall z \leq \overline{k})\neg A(\overline{n}, z)$ is provable in *Th*.

   So $(B(\overline{n}, \overline{k}) \wedge (\forall z \leq \overline{k})\neg A(\overline{n}, z))$ is provable in *Th*.

   So $\neg(B(\overline{n}, \overline{k}) \rightarrow \neg(\forall z \leq \overline{k})\neg A(\overline{n}, y))$ is provable in *Th*.

   But this is $\neg(B(\overline{n}, \overline{k}) \rightarrow (\exists z \leq \overline{k})A(\overline{n}, z))$. So $(B(\overline{n}, \overline{k}) \rightarrow (\exists z \leq \overline{k})A(\overline{n}, z))$ is refutable in *Th*.

   So $\forall y(B(\overline{n}, y) \rightarrow (\exists z \leq y)A(\overline{n}, z))$ is refutable in *Th*.

The Definability Theorem

Lastly, we have to observe that if any two disjoint Σ_1 relations are separable in *Th*, then all recursive relations are definable in *Th*.

Suppose that any two distinct Σ_1 relations are separable in *Th*, and let *R* be a recursive relation. Since *R* is recursive, both *R* and $\tilde{R}$ are Σ_1. Obviously, *R* and $\tilde{R}$ are disjoint. So *R* and $\tilde{R}$ are separable in *Th*, i.e., there is a formula $\mathsf{F}(\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_m)$ that *separates* relation *R* from relation $\tilde{R}$ in a theory *Th*, which means:

(1) If $\langle n_1, \ldots, n_m \rangle \in R$, then $\mathsf{F}(\overline{\mathsf{n}}_1, \ldots, \overline{\mathsf{n}}_m)$ is provable in *Th*.
(2) If $\langle n_1, \ldots, n_m \rangle \in \tilde{R}$, then $\mathsf{F}(\overline{\mathsf{n}}_1, \ldots, \overline{\mathsf{n}}_m)$ is refutable in *Th*.

(2) can be rewritten thus:

(2′) If $\langle n_1, \ldots, n_m \rangle \notin R$, then $\mathsf{F}(\overline{\mathsf{n}}_1, \ldots, \overline{\mathsf{n}}_m)$ is refutable in *Th*.

So, by definition of definability, *R* is definable in *Th*.

This completes the proofs of the four theorems that we needed to prove in order to prove that all recursive relations are definable in R.

The definability of recursive functions

There is one more task I wish to complete in this lesson, and that is to show that recursive *functions* are *strongly* definable in R. (Recall that that involves a third condition, (3), above.)

Theorem 2: All recursive functions are strongly definable in R.

Proof: For simplicity, we confine our attention to functions of one argument.

Suppose that *fun* is a recursive function of one argument. Since all recursive relations are definable in R (Theorem 1), there is a formula $\mathsf{F}(\mathsf{x}, \mathsf{y})$ that defines the relation $fun(x) = y$. This means that:

(i) If $fun(n) = m$, then $\mathsf{F}(\overline{\mathsf{n}}, \overline{\mathsf{m}})$ is provable in R.
(ii) If $fun(n) \neq m$, then $\mathsf{F}(\overline{\mathsf{n}}, \overline{\mathsf{m}})$ is refutable in R.

Let $G(x, y)$ abbreviate the formula $(F(x, y) \wedge \forall z(F(x, z) \rightarrow y \leq z))$. We will show that $G(x, y)$ strongly defines *fun*, i.e., that:

(1) If $fun(n) = m$, then $G(\overline{n}, \overline{m})$ is provable in R.
(2) If $fun(n) \neq k$, then $G(\overline{n}, \overline{k})$ is refutable in R.
(3) If $fun(n) = m$, then the sentence,
$\forall v(G(\overline{n}, v) \rightarrow v = \overline{m})$
is provable in R.

Suppose $fun(n) = m$.

(1) If $k < m$, then, by (ii), $F(\overline{n}, \overline{k})$ is refutable in R; so $(F(\overline{n}, \overline{k}) \rightarrow \overline{m} \leq \overline{k})$ is provable in R, and $(z = \overline{k} \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$ is provable. For instance, if $0 < m$, then $(F(\overline{n}, 0) \rightarrow \overline{m} \leq 0)$ is provable, and $(z = 0 \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$ is provable.

If $k = m$, then $\overline{m}$ and $\overline{k}$ are the same numeral, so that $\overline{m} = \overline{k}$ is provable and, by Ω_4, $\overline{m} \leq \overline{k}$ is provable; so $(F(\overline{n}, \overline{k}) \rightarrow \overline{m} \leq \overline{k})$ is provable, and $(z = \overline{m} \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$ is provable.

So each of $(z = 0 \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$, $(z = 0' \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$, …, $(z = \overline{m} \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$ is provable.
So $((z = 0 \vee z = 0' \vee \ldots \vee z = \overline{m}) \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$ is provable.

By Proposition 1, $(z \leq \overline{m} \rightarrow (z = 0 \vee z = 0' \vee \ldots \vee z = \overline{m}))$ is provable.

So $(z \leq \overline{m} \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$ is provable.

$(\overline{m} \leq z \rightarrow (F(\overline{n}, z) \rightarrow \overline{m} \leq z))$ is provable, by propositional logic.

So, by Ω_5, $(F(\overline{n}, z) \rightarrow \overline{m} \leq z)$ is provable, and by Generalization, $\forall z(F(\overline{n}, z) \rightarrow \overline{m} \leq z)$ is provable.

By (i), $F(\overline{n}, \overline{m})$ is provable in R.

So $(F(\overline{n}, \overline{m}) \wedge \forall z(F(\overline{n}, z) \rightarrow \overline{m} \leq z))$, i.e., $G(\overline{n}, \overline{m})$ is provable in R.

(2) Suppose $k \neq m$. Then, by (ii), $F(\overline{n}, \overline{k})$ is refutable in R.
So $(F(\overline{n}, \overline{k}) \wedge \forall z(F(\overline{n}, z) \rightarrow \overline{k} \leq z))$, i.e., $G(\overline{n}, \overline{k})$ is refutable in R.

(3) To show that $\forall v(G(\overline{n}, v) \rightarrow v = \overline{m})$ is provable, we will show:

(a) $(G(\overline{n}, v) \rightarrow v \leq \overline{m})$ is provable, and
(b) $(v \leq \overline{m} \rightarrow (G(\overline{n}, v) \rightarrow v = \overline{m}))$ is provable.

From (a) and (b), we derive, by propositional logic, $(G(\overline{n}, v) \rightarrow v = \overline{m})$, from which we obtain $\forall v(G(\overline{n}, v) \rightarrow v = \overline{m})$ by Generalization.

(a) Since $G(\overline{n}, v)$ is $(F(\overline{n}, v) \wedge \forall z(F(\overline{n}, z) \rightarrow v \leq z))$,
$(G(\overline{n}, v) \rightarrow \forall z(F(\overline{n}, z) \rightarrow v \leq z))$ is provable.
So $(G(\overline{n}, v) \rightarrow (F(\overline{n}, \overline{m}) \rightarrow v \leq \overline{m}))$ is provable.
By (i), $(F(\overline{n}, \overline{m})$ is provable.
So $(G(\overline{n}, v) \rightarrow v \leq \overline{m})$ is provable.

(b) If $k < m$, then by (2), $G(\overline{n}, \overline{k})$ is refutable; so $(G(\overline{n}, \overline{k}) \rightarrow \overline{k} = \overline{m})$ is provable.
If $k = m$, then $\overline{k} = \overline{m}$ is provable; so $(G(\overline{n}, \overline{k}) \rightarrow \overline{k} = \overline{m})$ is provable.
So, by reasoning in the by-now-familiar way (see (1) in the proof of this theorem), $(v \leq \overline{m} \rightarrow (G(\overline{n}, v) \rightarrow v = \overline{m}))$ is provable.

End of proof

Lesson 10: The Upper Diagonal Lemma and Some Consequences

Our objective is to prove another diagonal lemma. This other diagonal lemma will be of use in proving that no consistent extension of R is decidable. That in turn will lead fairly directly to a strong form of Gödel's First Incompleteness Theorem and to the Undecidability of First-order Logic.

The strong definability of the diagonal function

Proposition 1: For any n-ary function *fun*, if the relation $fun(x_1, x_2, \ldots, x_n) = x_{n+1}$ is Σ_1, then *fun* is recursive (by which I mean that *fun* is recursive in the sense of "recursive" that we defined for functions in Lesson 6). (Recall that "recursive" is short for "recursively decidable".)

> *Proof:* Suppose $fun(x_1, x_2, \ldots, x_n) = x_{n+1}$ is Σ_1. And suppose $\mathsf{F}(\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n, \mathsf{x}_{n+1})$ is a Σ_1-formula that expresses it. Then the complement of this relation, viz., the relation $fun(x_1, x_2, \ldots, x_n) \neq x_{n+1}$, is Σ_1 too, because it can be expressed with the following Σ-formula:
>
> $$\exists \mathsf{y}(\mathsf{F}(\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_n, \mathsf{y}) \wedge \mathsf{y} \neq \mathsf{x}_{n+1})$$

Say that E_n is the expression for which n is the Gödel number. So, recall, $diag(n) = m$ if and only if m is the code of $\forall \mathsf{v}(\mathsf{v} = \overline{\mathsf{n}} \rightarrow \mathsf{E}_n)$.

Recall that in Lesson 7, we found that *diag* is arithmetic. In fact, inspection of the formula that expresses it shows that it is Σ_1. (We cannot find a Σ_0-formula that expresses it, because we do not have a Σ_0-formula that expresses exponentiation.)

Refer back to Lesson 9 for the definition of *strongly definable function*.

Theorem: The diagonal function *diag* is strongly definable in R.

> *Proof:* As we have seen, *diag* is Σ_1. So by the above proposition, it is recursive. So by Theorem 2 of Lesson 9, it is strongly definable in R.

The upper diagonal lemma

Before proving our new diagonal lemma, we must prove the following proposition:

Proposition 2: If a one-place function *fun* is strongly definable in a theory *Th*, then for any formula $F(v)$ (in the language of the theory) there is a formula $H(v)$ such that for any number *n*, if *fun*(*n*) = *m*, then the following sentence is provable in *Th*:

$(H(\overline{n}) \leftrightarrow F(\overline{m}))$

Proof: Suppose $K(v, w)$ strongly defines *fun* in *Th*. We show that if *fun*(*n*) = *m*,

$(\exists w(K(\overline{n}, w) \wedge F(w)) \leftrightarrow F(\overline{m}))$

is provable in *Th*.

Suppose *fun*(*n*) = *m*.

1. Since $K(v, w)$ strongly defines *fun*, $K(\overline{n}, \overline{m})$ is provable in *Th*. So $(F(\overline{m}) \rightarrow (K(\overline{n}, \overline{m}) \wedge F(\overline{m}))$ is provable; so $(F(\overline{m}) \rightarrow \exists w(K(\overline{n}, w) \wedge F(w))$ is provable.

2. Since $K(v, w)$ strongly defines *fun*, $\forall v(K(\overline{n}, v) \rightarrow v = \overline{m})$ is provable in *Th*. Therefore, $\forall v((K(\overline{n}, v) \wedge F(v)) \rightarrow (v = \overline{m} \wedge F(v))$ is provable; so $\forall v((K(\overline{n}, v) \wedge F(v)) \rightarrow F(\overline{m}))$ is provable; so $(\exists w(K(\overline{n}, w) \wedge F(w)) \rightarrow F(\overline{m}))$ is provable.

End of proof

Definition: Where X is an expression (of the language of arithmetic), let $\overline{X}$ be the numeral denoting $\#(X)$, the Gödel number of X.

Definition: A sentence G is a *fixed point* of a formula $F(v)$ in a theory *Th* if and only if the sentence $(G \leftrightarrow F(\overline{G}))$ is provable in *Th*.

Definition: We say that one theory *Th1* is an *extension* of another theory *Th2* if and only if all of the theorems of *Th2* are also theorems of *Th1*, i.e., Con(*Th2*) $\subseteq$ Con(*Th1*). (Unless otherwise noted, we assume that an extension of a theory is in the same language as the theory.)

The Upper Diagonal Lemma: Every formula $\mathsf{F(v)}$ of $\mathcal{L}_A$ has a fixed point in any consistent extension *Th* of R.

Proof: Let $\mathsf{F(v)}$ be a formula in $\mathcal{L}_A$, and let *Th* be a consistent extension of R. Since *diag* is strongly definable in R, and hence in *Th*, we know by Proposition 2 that there is a formula $\mathsf{H(v)}$ such that for any number *n*, if *diag*(*n*) = *m*, then the following sentence is provable in *Th*:

$$(\mathsf{H}(\overline{n}) \leftrightarrow \mathsf{F}(\overline{m}))$$

Let *h* be the code of $\mathsf{H(v)}$, and suppose *diag*(*h*) = *k*. So:

$$(\mathsf{H}(\overline{h}) \leftrightarrow \mathsf{F}(\overline{k}))$$

is provable in *Th*. So

$$(\forall \mathsf{v}(\mathsf{v} = \overline{h} \rightarrow \mathsf{H(v)}) \leftrightarrow \mathsf{F}(\overline{k}))$$

is provable in *Th*. But *k* is the code of $\forall \mathsf{v}(\mathsf{v} = \overline{h} \rightarrow \mathsf{H(v)})$. So $\forall \mathsf{v}(\mathsf{v} = \overline{h} \rightarrow \mathsf{H(v)})$ is a fixed point for $\mathsf{F(v)}$.

Compare this result to the Lower Diagonal Lemma in Lesson 7. Here, rather than showing merely that *diag* is arithmetic, we show that, since *diag* is Σ_1, it is strongly definable in R. And instead of proving that the Gödel sentence for a set is true if and only if its code belongs to the set, we prove that the fixed point (a Gödel sentence) is provably (in *Th*) materially equivalent to the sentence that says that its code satisfies $\mathsf{F(v)}$.

The Upper Diagonal Lemma is not a consequence of the Lower Diagonal Lemma, since the Lower Diagonal Lemma does not deal with arbitrary consistent extensions of R.

But the Lower Diagonal Lemma is a consequence of the Upper Diagonal Lemma: Suppose *A* is arithmetic and $\mathsf{F(v)}$ expresses *A*. Since $\mathcal{N}$ (the set of truths of arithmetic) is an extension of R, by the Upper Diagonal Lemma, there is a sentence G such that $(\mathsf{G} \leftrightarrow \mathsf{F}(\overline{\mathsf{G}}))$ is provable in $\mathcal{N}$ and, hence true, so that G is true if and only if $\mathsf{F}(\overline{\mathsf{G}})$ is true. But $\mathsf{F}(\overline{\mathsf{G}})$ is true if and only if $\#(\mathsf{G}) \in A$. So G is true if and only if $\#(\mathsf{G}) \in A$.

The undecidability of consistent extensions of R.

Recall that if F is any formula, then $\#(\mathsf{F})$ is the Gödel number (code) of F.

Lemma: If *Th* is a consistent extension of R, then the set of codes of formulas provable in *Th* is not definable in *Th*.

> *Proof:* Suppose *Th* is a consistent extension of R. Let *P* be the set of codes of formulas provable in *Th*, and suppose, for a reductio, that *P* is definable in *Th*. Suppose that $\mathsf{H}(\mathsf{v})$ defines *P* in *Th*. So:
>
> > If $n \in P$, then $\mathsf{H}(\overline{\mathsf{n}})$ is provable in *Th*.
> > If $n \notin P$, then $\mathsf{H}(\overline{\mathsf{n}})$ is refutable in *Th*, i.e., $\neg\mathsf{H}(\overline{\mathsf{n}})$ is provable in *Th*.
>
> By the Upper Diagonal Lemma, there is a fixed point for $\neg\mathsf{H}(\mathsf{v})$, i.e., a sentence G such that
>
> > $(\mathsf{G} \leftrightarrow \neg\mathsf{H}(\overline{\mathsf{G}}))$
>
> is provable in *Th*. We show both that G is provable and not provable (in *Th*).
>
> G *is not provable:* Suppose, for a reductio, that G is provable. Then $\#(\mathsf{G}) \in P$. So $\mathsf{H}(\overline{\mathsf{G}})$ is provable. So by the provability of the above biconditional, $\neg\mathsf{G}$ is provable. So, since *Th* is consistent, G is not provable.
>
> G *is provable:* Suppose, for a reductio, that G is not provable. Then $\#(\mathsf{G}) \notin P$. So $\neg\mathsf{H}(\overline{\mathsf{G}})$ is provable. So by the provability of the above biconditional, G is provable.
>
> Contradiction!

Theorem (the undecidability of consistent extensions of R): If *Th* is a consistent extension of R, then the set of codes of theorems of *Th* is not recursive (so that, by Church's thesis, the set of theorems of *Th* is not decidable).

> *Proof:* Suppose that *Th* is a consistent extension of R. By the above Lemma, the set of codes of theorems of *Th* is not definable in *Th*. But Theorem 1 of Lesson 9 states that all recursive relations are definable in R. So all recursive relations are definable in any extension of R. So the set of codes of theorems of *Th* is not recursive.

Note: We call this result the "undecidability of consistent extensions of R", although strictly speaking what is undecidable is not the set *Th* but the set Con(*Th*). In general, we will say that a *theory* is undecidable when what we mean is that the set of theorems of the theory is undecidable. (As I said at the beginning of Lesson 8, some people reserve the term "theory" for sets of sentences closed under first-order consequence.) Later, we will say that *first-order logic* is undecidable when what we mean is that set of theorems of QL is undecidable.

Advisement: Nothing in subsequent lessons will depend on any of the rest of this lesson. The rest of this lesson is here for its intrinsic interest.

Tarski Undefinability Revisited:

I pause now to prove a second version of the Tarski Undefinability Theorem. We will have no further use for this, but it is the form of the theorem that one usually encounters in the literature (e.g., the literature on the liar paradox).

Definition: Say that $\mathsf{T}(\mathsf{v})$ is a *truth-predicate* for a theory *Th* if and only if for every sentence S, the sentence $(\mathsf{S} \leftrightarrow \mathsf{T}(\overline{\mathsf{S}}))$ is provable in *Th*.

Tarski's Undefinability Theorem (upper version): If *Th* is any consistent extension of R, then there is no truth-predicate for *Th*.

> *Proof:* Suppose, for a reductio, that $\mathsf{T}(\mathsf{v})$ is a truth-predicate for *Th*, a formally consistent extension of R. By the Upper Diagonal Lemma, $\neg\mathsf{T}(\mathsf{v})$ has a fixed point, call it G. So $(\mathsf{G} \leftrightarrow \neg\mathsf{T}(\overline{\mathsf{G}}))$ is provable in *Th*. But since $\mathsf{T}(\mathsf{v})$ is a truth predicate, $(\mathsf{G} \leftrightarrow \mathsf{T}(\overline{\mathsf{G}}))$ is provable in *Th*. So $\mathsf{T}(\overline{\mathsf{G}}) \leftrightarrow \neg\mathsf{T}(\overline{\mathsf{G}})$ is provable in *Th*. So *Th* is formally inconsistent, contrary to assumption.

Corollary: There is no truth predicate for $\mathcal{N}$, the set of truths of arithmetic.

> *Proof:* $\mathcal{N}$ is a consistent extension of R.

The decidability of effectively enumerable, complete theories: If *Th* is complete and the set of codes of theorems of *Th* is recursively enumerable, then the set of codes of theorems of *Th* is recursive.

Recall that by a *complete* theory, we mean a theory such that for every sentence P (in the language of the theory), either P or $\neg\mathsf{P}$ is a theorem.

Nota bene: We do not say that in a complete theory every *formula* or its negation is a theorem. However, if a theory is complete, then for every formula F, if $\forall v_1 \ldots \forall_n \mathsf{F}$ is a sentence, then either $\forall v_1 \ldots \forall_n \mathsf{F}$ or $\neg \forall v_1 \ldots \forall_n \mathsf{F}$ is a theorem.

We show that if a theory is complete, and the set of codes of theorems is recursively enumerable (i.e., Σ_1), then the set of codes of theorems is recursive (i.e., recursively decidable). By Church's thesis, it follows that if a theory is complete and its theorems are *effectively* enumerable, then the set of theorems is decidable.

Suppose that *Th* is complete and *P*, the set of codes of theorems of *Th*, is Σ_1. To obtain our result, all we have to do is show that $\tilde{P}$ is Σ_1. ($\tilde{P}$ includes numbers that are codes of expressions that are not even formulas, as well as codes of formulas that are not theorems.)

Case 1: Th is inconsistent. $\tilde{P}$ is the empty set and therefore Σ_1.

Case 2: Th is consistent.

Say that a formula P is the *opposite* of a formula Q if and only if either $\mathsf{Q} = \neg\mathsf{P}$ or $\mathsf{P} = \neg\mathsf{Q}$. Say that a sentence P is a *universal closure* of a formula Q if and only if: if $v_1, \ldots, v_n$ are the variables in Q, then $\mathsf{P} = \forall v_1 \ldots \forall v_n \mathsf{Q}$. (For simplicity, we do not assume that all of $v_1, \ldots, v_n$ are *free* in Q. So some of the quantifiers in the universal closure may be vacuous.) I take for granted that the following relations are Σ_1:

x is a code of an opposite of the formula of which *y* is the code
x is a code of a universal closure of the formula of which *y* is the code

Let us abbreviate the Σ_1 formulas that express these relations as $\mathsf{Opp}(\mathsf{x}, \mathsf{y})$ and $\mathsf{UC}(\mathsf{x}, \mathsf{y})$, respectively.

Recall from the end of Lesson 8 that the set of codes that are not codes of well-formed formulas is Σ_1. Let $\mathsf{NonForm}(\mathsf{x})$ abbreviate the Σ_1-formula that expresses that set.

Since the set of codes of theorems of *Th* is recursively enumerable, there is a Σ_1 formula Prov(v) that expresses it. Since *Th* is consistent and complete, the following is a Σ-formula that expresses the set of codes of nonformulas and *un*provable formulas, $\tilde{P}$.

(NonForm(x) ∨ ∃y(Prov(y) ∧ (Opp(x, y) ∨ ∃z(UC(z, x) ∧ Opp(y, z)))))

For example, if F(v) is not provable, then either its opposite is provable or ¬∀vF(v) is provable.

Reasoning informally, the truth of this theorem should be clear. If *Th* is a complete theory and its theorems are enumerable, then it is decidable whether any formula is a theorem of *Th*, because for every formula, either it or its negation or the negation of its universal closure is bound to show up in the enumeration (and if the negation of the universal closure of F shows up, and the theory is consistent, then we know that F is not going to show up).

In Lesson 8, we defined the concept of a *correctly axiomatizable* theory. We are not concerned with correctness (truth) any more. So we will say that a theory is *axiomatizable* if and only if there is a decidable set of formulas *A* such that all of the members of the set are consequences of *A*.

Gödel's First Incompleteness Theorem (fourth formulation): No consistent, complete extension of R is axiomatizable.

Proof: Suppose, for a reductio, that *Th* is a consistent, complete, axiomatizable extension of R. By analogy with the arithmetization of proof in P.A. in Lesson 8, the set of codes of theorems of *Th* is Σ_1. By the decidability of effectively enumerable, complete theories, the set codes of theorems of *Th* is recursive. By the undecidability of consistent extensions of R, the set of codes of theorems of *Th* is *not* recursive. Contradiction!

This of course entails that $\mathcal{N}$ is not axiomatizable (and thus not correctly axiomatizable, our third formulation, in Lesson 8), because $\mathcal{N}$ is a consistent, complete extension of R.

The virtue of this fourth formulation and its proof, in comparison with our earlier proofs, is that we did not have to assume that R is true, only that it is consistent. And R is a *very* weak theory. (Just look at it!)

Lesson 11: The Undecidability of First-order Logic

Another theory that will be of use to us is the theory called simply Q. Q is just the first nine axioms of P.A. (so, P.A. minus the induction scheme), which I repeat here for convenience:

N1: $(x' = y' \rightarrow x = y)$
N2: $\neg 0 = x'$
N3: $(x + 0) = x$
N4: $(x + y') = (x + y)'$
N5: $(x \bullet 0) = 0$
N6: $(x \bullet y') = ((x \bullet y) + x)$
N7: $(x \leq 0 \leftrightarrow x = 0)$
N8: $(x \leq y' \leftrightarrow (x \leq y \vee x = y'))$
N9: $(x \leq y \vee y \leq x)$

Let A be the universal closure of the conjunction of these nine formulas And suppose, for a reductio, that first-order logic is decidable (i.e., it is decidable whether any given formula is a theorem of QL, which means, by soundness and completeness, that it is also decidable whether any given formula is first-order valid). Then, assuming that the language of QL contains $\mathcal{L}_A$, for any formula S of $\mathcal{L}_A$, it is decidable whether $(\mathsf{A} \rightarrow \mathsf{S})$ is a theorem of first-order logic. But $(\mathsf{A} \rightarrow \mathsf{S})$ is a theorem of first order logic if and only if S is a theorem of Q. So Q is decidable. But we already know that any consistent extension of R is undecidable. So suppose we can show that Q is a consistent extension of R Then Q is undecidable. Contradiction!

So if we can show that Q is a consistent extension of R, it will follow that first-order logic is not decidable. So our first order of business is to show that Q is a consistent extension of R We will simply *assume* that Q is consistent; but we need to show that Q is an extension of R

For convenience, I also repeat the specification of the axioms of R:

Ω_1: All sentences $(\overline{m} + \overline{n}) = \overline{k}$, where $m + n = k$.
Ω_2: All sentences $(\overline{m} \bullet \overline{n}) = \overline{k}$, where $m \times n = k$.
Ω_3: All sentences $\overline{m} \neq \overline{n}$, where m and n are distinct numbers.
Ω_4: For each n, $v_* \leq \overline{n} \leftrightarrow (v_* = 0 \vee v_* = 0' \vee \ldots \vee v_* = \overline{n})$.
Ω_5: For each n, $v_* \leq \overline{n} \vee \overline{n} \leq v_*$.

Theorem: Q is an extension of R That is, Con(R) $\subseteq$ Con(Q).

Proof: We need to prove that for each of the five kinds of axioms of R each axiom of that kind is a theorem of Q. That will ensure that every theorem of R is also a theorem of Q.

Ω_1: By N3, for each *m*, we have $(\overline{m} + 0) = \overline{m}$. By N4, we have $(\overline{m} + 0') = (\overline{m} + 0)'$. So, by substitution of identicals, we have $(\overline{m} + 0') = \overline{m}'$. In other words, $(\overline{m} + 1) = \overline{m+1}$. By N4, $(\overline{m} + 0'') = (\overline{m} + 0')'$. By substitution of identicals, $(\overline{m} + 0'') = \overline{m+1}'$, i.e., $(\overline{m} + 2) = \overline{m+2}$. Similarly, $(\overline{m} + 3) = \overline{m+3}$, …, $(\overline{m} + \overline{n}) = \overline{m+n}$, … This gives us every axiom of type Ω_1.

Ω_2: By N5, we have, for each *m*, $(\overline{m} \bullet 0) = 0$. By N6, we have $(\overline{m} \bullet 1) = ((\overline{m} \bullet 0) + \overline{m})$. So, by substitution of identicals, we have $(\overline{m} \bullet 1) = (0 + \overline{m})$. By the previous paragraph, we have $(0 + \overline{m}) = \overline{0+m}$. So we have $(\overline{m} \bullet 1) = \overline{0+m}$. So we have $(\overline{m} \bullet 1) = \overline{m \times 1}$. By N6 again, we have $(\overline{m} \bullet 1') = ((\overline{m} \bullet 1) + \overline{m})$. So, by substitution of identicals, we have $(\overline{m} \bullet 1') = (\overline{m \times 1} + \overline{m})$, i.e., $(\overline{m} \bullet 2) = (\overline{m} + \overline{m})$. By the previous paragraph, we have $(\overline{m} + \overline{m}) = \overline{m \times 2}$. So we have $(\overline{m} \bullet 2) = \overline{m \times 2}$. And so on, for each *n*, we have $(\overline{m} \bullet \overline{n}) = \overline{m \times n}$. This gives us every axiom of type Ω_2.

Ω_3: By N1, we have, for every *m* and *n*, $(\overline{m+1} = \overline{n+1} \rightarrow \overline{m} = \overline{n})$. By N2, for any *positive n*, $0 \neq \overline{n}$. So $\overline{0+1} \neq \overline{n+1}$, i.e., $1 \neq \overline{n+1}$. So $\overline{1+1} \neq \overline{n+2}$, i.e, $2 \neq \overline{n+2}$, and so on. This gives us every axiom of type Ω_3.

Ω_4: By N7 we have, $(v_* \leq 0 \leftrightarrow v_* = 0)$. Suppose, as an induction hypothesis, that $(v_* \leq \overline{n} \leftrightarrow (v_* = 0 \vee \ldots \vee v_* = \overline{n}))$. By N8, we have $(v_* \leq \overline{n+1} \leftrightarrow (v_* \leq \overline{n} \vee v_* = \overline{n+1})$. From these last two formulas, we derive: $v_* \leq \overline{n+1} \leftrightarrow (v_* = 0 \vee v_* = 0' \vee \ldots \vee v_* = \overline{n+1})$.

Ω_5: These are all consequences of N9.

End of proof.

The undecidability of first-order logic

Theorem (the undecidability of first-order logic): The set of codes of formulas of $\mathcal{L}_A$ that are first-order valid is not recursive. (And so, by Church's thesis, the set of valid formulas of the language of arithmetic is not decidable.)

Proof: (For the basic idea, see above. We now give a precise proof.)

Let V be the set of codes of first-order valid formulas of $\mathcal{L}_A$, and let $\tilde{V}$ be the complement of that set (the union of the set codes of nonformulas and the set of codes of nonvalid formulas). Suppose, for a reductio that V is recursive. So there is a Σ_1-formula $\mathsf{Val(x)}$ that expresses V and a Σ_1-formula $\mathsf{NonVal(x)}$ that expresses $\tilde{V}$.

Let A be the universal closure of the conjunction of the axioms of Q (i.e., the result of adding $\forall v_* \forall v_{**}$ to that conjunction), and let a be the code for A.

Then we can define a Σ-formula that expresses the set of codes of formulas S such that $(\mathsf{A} \rightarrow \mathsf{S})$ is first-order valid. Here it is:

$$\exists y(\mathsf{Val}(y) \wedge \mathsf{Concat}_5(2, \overline{a}, 8, x, 3, y)).$$

And we can define a Σ-formula that expresses the set of codes of expressions S such that $(\mathsf{A} \rightarrow \mathsf{S})$ is *not* first-order valid (either not a formula at all, or not a valid formula). Here it is:

$$\exists y(\mathsf{NonVal}(y) \wedge \mathsf{Concat}_5(2, \overline{a}, 8, x, 3, y)).$$

But these two formulas express the set of codes of theorems of Q and the complement of that set, respectively. So by Smullyan's Theorem, we can find Σ_1-formulas that express the same sets. So the set of codes of theorems of Q is recursive.

But by the above theorem, Q is a consistent extension of R (since we're assuming that Q is consistent). So by the main result of Lesson 10, the set of theorems of Q is *not* recursive. Contradiction!

End of proof.

First note: We have not shown that every first-order language is undecidable, i.e., that the set of valid sentences of any first-order language is undecidable. We have only shown that the set of first-order valid sentences of $\mathcal{L}_A$ is undecidable. And in fact, the valid formulas of a first-order language containing exclusively *monadic* predicates *is* decidable. (That fact requires a proof that we will not go through.) But consider a language $\mathcal{L}$ that is similar to $\mathcal{L}_A$ to the extent that it contains at least two dyadic predicates (like "$=$" and "$\leq$"), and at least one one-place function symbol (like "$'$") and at least two two-place function symbols (like "$+$", "$\bullet$"). It is clear that if we had an algorithm for deciding whether a formula in $\mathcal{L}$ was first-order valid, then we would have an algorithm for deciding whether a formula in $\mathcal{L}_A$ was valid, which, as we have just seen, we do not have. So we may conclude that first-order validity is not decidable even in this broader class of languages. Likewise, the set of first-order valid sentences of any language with a more extensive vocabulary will be undecidable, for it were decidable, then the first-order valid sentences of this "sublanguage" would be decidable.

Second note: The set of valid arguments is not decidable, for if it were then the set of valid arguments with finitely many premises would be decidable, and if that were decidable, then the set of first-order valid conditionals would be decidable, and if that were decidable, we could likewise show that the set of theorems of Q was decidable.

Third note: Although first-order logic is undecidable (i.e., again, the set of first-order valid formulas is undecidable), for any invalid argument we *can* demonstrate that it is invalid. We do so by describing a structure in which the premises are true and the conclusion is not true. First-order logic is undecidable because we have no algorithm for finding such counterexamples.

Thought exercise: The theorems of first-order logic (which, by, soundness and completeness are the first-order valid formulas) are effectively enumerable (by the arithmetization of proof). So first-order logic would be decidable if we could effectively enumerate the set comprising nonformulas and formulas that are not valid. We cannot do this, but why not? If you were to try to find a Σ_1-formula that expresses the codes of nonprovable formulas, where would you encounter a problem?

Lesson 12: Gödel's Second Incompleteness Theorem

What Gödel's Second Incompleteness Theorem says is that Peano Arithmetic (P.A.) cannot prove its own consistency. This result can be generalized by paying attention to which properties of P.A. we actually use in the proof. Then we can say of every theory that has those properties that it cannot prove its own consistency. But for simplicity, I will confine attention to P.A. I am ripping this presentation pretty much straight out of Smullyan, pp. 106-109, but I am adding some material that I took from George Boolos, *The Logic of Provability*, Cambridge University Press, 1994.

Provability predicates

Recall that if X is an expression, then $\overline{X}$ is the numeral denoting the code of that expression, $\#(X)$.

Definition: A formula $P(x)$ is a *provability predicate* for a theory *Th* if and only if the following three conditions hold:

P1: If X is provable in *Th*, then $P(\overline{X})$ is provable in *Th*.
P2: $(P(\overline{X \rightarrow Y}) \rightarrow (P(\overline{X}) \rightarrow P(\overline{Y})))$ is provable in *Th*.
P3: $P(\overline{X}) \rightarrow P(\overline{P(\overline{X})})$ is provable in *Th*.

Theorem: If $P(x)$ is a provability predicate for *Th*, then the following three facts hold:

P4: If $(X \rightarrow Y)$ is provable in *Th*, then $(P(\overline{X}) \rightarrow P(\overline{Y}))$ is provable in *Th*.
P5: If $(X \rightarrow (Y \rightarrow Z))$ is provable in *Th*, then $(P(\overline{X}) \rightarrow (P(\overline{Y}) \rightarrow P(\overline{Z})))$ is provable in *Th*.
P6: If $(X \rightarrow (P(\overline{X}) \rightarrow Y))$ is provable in *Th*, then $(P(\overline{X}) \rightarrow P(\overline{Y}))$ is provable in *Th*.

Proof:

P4: Suppose $(X \rightarrow Y)$ is provable.
By P1, $P(\overline{X \rightarrow Y})$ is provable.
By P2 and Modus Ponens, $(P(\overline{X}) \rightarrow P(\overline{Y}))$ is provable.
P5: Suppose $(X \rightarrow (Y \rightarrow Z))$ is provable.
By P4, $(P(\overline{X}) \rightarrow P(\overline{Y \rightarrow Z}))$ is provable.
By P2, $(P(\overline{Y \rightarrow Z}) \rightarrow (P(\overline{Y}) \rightarrow P(\overline{Z})))$ is provable.

So, by propositional logic, $(P(\overline{X}) \rightarrow (P(\overline{Y}) \rightarrow P(\overline{Z})))$ is provable.

P6: Suppose $(X \rightarrow (P(\overline{X}) \rightarrow Y))$ is provable.

By P5, $(P(\overline{X}) \rightarrow (P(\overline{P(\overline{X})}) \rightarrow P(\overline{Y}))$ is provable.

By P3, $P(\overline{X}) \rightarrow P(\overline{P(\overline{X})})$ is provable.
So, by propositional logic, $(P(\overline{X}) \rightarrow P(\overline{Y}))$ is provable.

Theorem: The Σ_1-formula $Prov(x)$ that expresses the set of codes of formulas provable in P.A. is a provability predicate for P.A.

Note: Is there a Σ_1-formula that expresses the set of codes of formulas provable in P.A.? Yes, in Lesson 8 we showed that there is a Σ-formula that expresses the codes of formulas provable in P.A, which means, by Smullyan's theorem, that there is a Σ_1-formula that expresses it.

Proof: We will prove only that condition P1 and will sketch a proof that P2 holds. For a sketch of a proof that P3 holds (in the context of a different system of Gödel numbering), see George Boolos, *The Logic of Provability*, *op. cit*. For a detailed proof, one must apparently go to David Hilbert and Paul Bernays, *Grundlagen der Mathematik*, vol.1, 1934, vol. 2, 1939 (although I have never tried that myself).

Since $Prov(x)$ is Σ_1, there is some Σ_0-formula $Pf(x, y)$ such that $Prov(x)$ is $\exists y Pf(x, y)$. (Formerly, we said that $Prov(x)$ was $\exists y(Pf(y) \wedge x \text{ In } y)$, but now I will abbreviate the latter.) Since P.A. extends Q (by the addition of the induction axioms) and Q extends R.A. (Lesson 11) and R.A. is Σ_0-complete (Lesson 9), P.A. proves every true Σ_0-sentence.

P1: Suppose X is provable in P.A. In that case $Prov(\overline{X})$ is true. Since $Prov(\overline{X})$, i.e., $\exists y Pf(\overline{X}, y)$, is true, there is some number *n* such that $Pf(\overline{X}, \overline{n})$ is true. Since P.A. proves every true Σ_0-sentence, $Pf(\overline{X}, \overline{n})$ is provable in P.A. Therefore $\exists y Pf(\overline{X}, y)$, i.e, $Prov(\overline{X})$, is provable in P.A.

P2: In other words, we want to show that the following is provable in P.A.:

(i) $(\exists z Pf(\overline{X \rightarrow Y}, z) \rightarrow (\exists z Pf(\overline{X}, z) \rightarrow \exists z Pf(\overline{Y}, z)))$

To see that (i) is provable, observe, first of all, that it is true. Where $\overline{m}\delta$ is the numeral that denotes the code of a proof of $X \rightarrow Y$ and n is the numeral that denotes the code of a proof of X, a numeral that denotes the code of a proof of Y will be

$\overline{m}\,\overline{n}\,\overline{Y}\delta$. (For notation, see Lesson 5). The following Σ_0-formula expresses the relation between the proof of $X \rightarrow Y$, the proof of X and the proof of Y.

$$\exists m \leq z(z = m\delta \wedge o = mn\overline{Y}\delta)$$

Abbreviate this as $\mathsf{ConP}(z, n, o)$. The following sentences are provable in P.A. (we assume without proof):

$$\forall z \forall z'(\mathsf{Pf}(\overline{X \rightarrow Y}, z) \rightarrow \exists z'' \mathsf{ConP}(z, z', z''))$$

$$\forall z \forall z' \forall z''(\mathsf{Pf}(\overline{X \rightarrow Y}, z) \wedge \mathsf{Pf}(\overline{X}, z') \wedge \mathsf{ConP}(z, z', z'')) \rightarrow \mathsf{Pf}(\overline{Y}, z'')).$$

From these two sentence, it follows by logic that (i) is provable in P.A.

Comment on P3: The proof of this requires some instances of the induction scheme of P.A. So we cannot get by with appealing to the Σ_0-completeness of R.A., as we did in proving that $\mathsf{Prov}(x)$ satisfies P1 and P2.

The unprovability of consistency

We will assume that P.A. is consistent. We assume that $\mathsf{Prov}(x)$ is the Σ_1-formula that expresses the set of codes of theorems of P.A.

Let $\perp$ abbreviate some false sentence in the language of arithemetic, such as $0 = 1$. If P.A. is inconsistent, then every formula in the language of arithmetic is a theorem, including $\perp$. So if we can prove that $\perp$ is *not* provable in P.A., then we will prove that P.A. is consistent. So we can think of the statement "$\perp$ is not provable in P.A." as asserting the consistency of P.A. Further, since $\mathsf{Prov}(x)$ expresses the set of codes of formulas provable in P.A., if the sentence $\neg\mathsf{Prov}(\overline{\perp})$ is provable in P.A., then P.A. can be said to prove *its own* consistency. We will see that it does not do that.

Lemma 1: No fixed point in P.A. for $\neg\mathsf{Prov}(x)$ is provable in P.A.

Proof: Let G be a fixed point for $\neg\mathsf{Prov}(x)$. So $(G \leftrightarrow \neg\mathsf{Prov}(\overline{G}))$ is provable in P.A. Suppose, for a reductio, that G is provable in P.A. Then, since $\mathsf{Prov}(x)$ is a provability predicate for P.A., it follows, by property P1 of provability predicates, that $\mathsf{Prov}(\overline{G})$ is provable in P.A. But since $(G \leftrightarrow \neg\mathsf{Prov}(\overline{G}))$ is provable in P.A., $\neg\mathsf{Prov}(\overline{G})$ is provable in P.A. But P.A. is consistent. Contradiction!

Lemma 2: If G is a fixed point for ¬Prov(x), then (¬Prov($\overline{\perp}$) → G) is provable in P.A.

Proof: Suppose G is a fixed point for ¬Prov(x).
So (i) (G ↔ ¬Prov($\overline{G}$)) is provable in P.A.
Since ⊥ is refutable in P.A., (ii) (¬Prov($\overline{G}$) ↔ (Prov($\overline{G}$) → ⊥)) is provable in P.A.
By (i) and (ii), (iii) (G → (Prov($\overline{G}$) → ⊥)) is provable in P.A.
By (iii) and property P6 of provability predicates, (iv) (Prov($\overline{G}$) → Prov($\overline{\perp}$)) is provable in P.A.
By (iv), (¬Prov($\overline{\perp}$) → ¬Prov($\overline{G}$)) is provable in P.A.
By (i) and (iv), (¬Prov($\overline{\perp}$) → G) is provable in P.A.

Gödel's Second Incompleteness Theorem: The sentence ¬Prov($\overline{\perp}$) is not provable in P.A. (In other words, P.A. does not prove its own consistency.)

Proof: P.A. is a consistent extension of R.A. So by the Upper Diagonal Lemma, there is a fixed point G for ¬Prov(x). So:

(G ↔ ¬Prov($\overline{G}$))

is provable in P.A. By Lemma 2, (¬Prov($\overline{\perp}$) → G) is provable in P.A. But by Lemma 1, G is not provable in P.A. So ¬Prov($\overline{\perp}$) is not provable in P.A.

Note 1: This result does *not* say that we cannot prove the consistency of Peano Arithmetic. All it says is that Peano Arithmetic does not prove its *own* consistency. We *can* prove the consistency of Peano Arithmetic. We specify a structure for the language of arithmetic and then show that all of the axioms of P.A. are true in that structure. Of course, in doing this, we will have to take for granted a theory of truth for the language of arithmetic, and if the structure we choose is the "intended interpretation" of the language of arithmetic, then to show that each of the axioms is true in the structure, we will have to take for granted some facts about natural numbers.

Note 2: The more general statement that can be proved in the same way is that if *Th* is consistent and has a provability predicate and *diag* is strongly definable in *Th*, then *Th* does not prove its own consistency.

Lesson 13: Second-order Logic

In second-order logic, we have variables that hold the place of predicates and function symbols and we can bind those variables with quantifiers. This allows us to "say" things that we could not say otherwise. The quantifiers that bind these variables are called *second-order quantifiers*. The languages that contain second-order quantifiers are called *second-order languages*. Second-order *logic* is the logic of second-order languages (a logic I will define below).

For example, we can state Leibniz's law, the identity of indiscernibles.

$$\forall x \forall y(\forall F(F(x) \leftrightarrow F(y)) \rightarrow x = y)$$

For another example, instead of having infinitely many instances of the Peano induction scheme, we can have a single sentence — the second order Peano induction scheme — that says everything that is said by the formulas in that infinite collection of formulas:

$$\forall F(F(0) \rightarrow (\forall x(F(x) \rightarrow F(x')) \rightarrow \forall x F(x)))$$

Suppose we take this sentence and replace the remaining nonlogical constants with appropriate variables and existentially quantify. This gives us:

(i) $\exists z \exists g \forall F(F(z) \rightarrow (\forall x(F(x) \rightarrow F(g(x))) \rightarrow \forall x F(x)))$

This sentence (as well as the second order Peano induction scheme) is true in a structure if and only if the the domain of that structure is countable (either finite or denumerably infinite). (For proof, see the "axiom of enumerability" in Boolos and Jeffrey, chapter 18.) In Lesson 4, I showed you a set of sentences that can all be true only in an infinite domain:

(ii) $\forall x \neg Rxx$
(iii) $\forall x \exists y Rxy$
(iv) $\forall x \forall y \forall z((Rxy \wedge Ryz) \rightarrow Rxz)$

So the set of sentences containing (i)-(iv) is satisfiable only in structures having a domain that is at least denumerably infinite.

We can even write sentences that violate the Löwenheim-Skolem Theorem. For example, the negation of sentence (i) will be true only in domains having a cardinality greater than that of the set of natural numbers (nondenumerably infinite).

Also, there seem to be ordinary sentences of English the meanings of which might best be expressed using second-order quantifiers. For example:

Some critics admire only one another.
$\exists F((\forall x(F(x) \rightarrow x \text{ is a critic}) \wedge \forall y(\exists z(F(z) \wedge z \text{ admires } y) \rightarrow F(y)))$

Now, one question you might have right away is: Why can't we do the same work with sets? For example, why couldn't we express the identity of indiscernibles as follows:

$\forall x \forall y(\forall s(x \in s \leftrightarrow y \in s) \rightarrow x = y)$?

Well, maybe for some purposes we could do that (for example for purposes of expressing the sentence about critics). But for other purposes we cannot. Compare the following two sentences, one of which uses second-order quantification and the other of which uses set membership:

(i) $\exists F \exists x F(x)$
(ii) $\exists s \exists x\ x \in s$

These two sentences fail to be equivalent. We cannot say that they are true in exactly the same structures. The reason is that "$\in$" is a vocabulary item that can be differently interpreted in different structures. So while (i) is (as we will see) a second-order valid sentence, (ii) is not.

Grammar

Let's suppose we have a second-order language, $\mathcal{L}^2$. I won't bother to write out the definition of well-formed formula or sentence for $\mathcal{L}^2$. I'll assume that you can recognize a formula of $\mathcal{L}^2$ well enough by understanding that we can have variables in place of function symbols and predicates and have quantifiers that bind those variables. These variables will be called *predicate variables* and *function variables*. The things that we formerly called "predicates" will now be called *predicate constants*, and the things we formerly called "function symbols" will now be called function symbols, as before, or *function constants*. The things that we formerly called "variables", will now be called *individual variables.*

We will adopt one new notational convention. We will put superscripts on predicate variables indicating their adicity. So F^n is a variable for which we substitute *n*-ary predicates. And f^n is a variable for which we substitute functions symbols for functions taking *n* arguments.

For simplicity I will go on assuming that the only logical constants are $\neg$, $\rightarrow$ and $\forall$ and that the other familiar logical constants are used to write abbreviations.

Semantics

As for first-order languages, a structure for a second-order language is a pair $\mathfrak{M} = \langle \mathsf{D}, \Sigma \rangle$ consisting of a domain D of objects and an assignment Σ. Just as before, Σ assigns to each individual constant a member of D, to each *n*-ary predicate an *n*-ary relation on members of D, and to each *n*-ary function symbol a function of *n* arguments on D.

But now we have to extend the concept of a variable assignment to include assignments to predicate and function variables. So a variable assignment in a structure $\mathfrak{M}$ is a function *g* such that:

(i) for each individual variable v, $g(\mathsf{v}) \in \mathsf{D}$, and
(ii) for each predicate variable F^n, $g(\mathsf{F}^n)$ is a set of *n*-tuples of members of D, and
(iii) for each function variable f^n, $g(\mathsf{f}^n)$ is a function of *n* arguments having the set of *n*-tuples of members of D as its domain and D as its range.

We assume that functions are *total*. That is, each *n*-ary function yields a value for every *n*-tuple of members of D.

We also need to have the concept of a *variant* of a variable assignment that allows variations on the assignments to predicate variables and function variables as well as variations on the assignments to individual variables. Thus:

$g[\mathsf{v}/o]$ is the variable assignment just like *g* except that it assigns *o* to v.
$g[\mathsf{F}^n/R]$ is the variable assignment just like *g* except that it assigns the set of *n*-tuples *R* to F^n.
$g[\mathsf{f}^n/fun]$ is the variable assignment just like *g* except that it assigns the function of *n* arguments *fun* to f^n.

In terms of g and Σ, we define a *term* assignment much as before. However, we now need a broader definition of *term*. On the first page of Lesson 5 I gave you a definition of *term* that included terms formed from function symbols. Take that as your definition of *individual term*, but allow "f" in that definition to include function variables. The *terms* are then such individual terms as well as predicate variables and predicate constants.

Where g is a variable assignment in $\mathfrak{M}$ and Σ is an assignment for $\mathfrak{M}$, and t is a term of $\mathcal{L}^2$, h is a *term* assignment in $\mathfrak{M}$ if and only if for all terms t of $\mathcal{L}^2$, either:

(i) t is an individual variable and $h(\mathsf{t}) = g(\mathsf{t})$, or
(ii) t is an individual constant and $h(\mathsf{t}) = \Sigma(\mathsf{t})$, or
(iii) t is a function variable and $h(\mathsf{t}) = g(\mathsf{t})$, or
(iv) t is a function constant and $h(\mathsf{t}) = \Sigma(\mathsf{t})$, or
(iv) for some individual terms $\mathsf{t}_1, \mathsf{t}_2, \ldots \mathsf{t}_n$ and some function variable or constant f, and some function *fun* of n arguments, $\mathsf{t} = \mathsf{f}(\mathsf{t}_1, \mathsf{t}_2, \ldots \mathsf{t}_n)$, and $h(\mathsf{f}) =$ *fun* and $h(\mathsf{t}) =$ *fun*$(h(\mathsf{t}_1), h(\mathsf{t}_2), \ldots, h(\mathsf{t}_n))$, or
(v) t is a predicate variable and $h(\mathsf{t}) = g(\mathsf{t})$, or
(vi) t is a predicate constant and $h(\mathsf{t}) = \Sigma(\mathsf{t})$.

Next, we define satisfaction in a structure by a variable assignment in the usual way:

(i) Where R is either a predicate variable or predicate constant and $\mathsf{t}_1, \mathsf{t}_2, \ldots, \mathsf{t}_n$ individual terms, g satisfies $\mathsf{R}(\mathsf{t}_1, \mathsf{t}_2, \ldots, \mathsf{t}_n)$ in $\mathfrak{M}$ if and only if $\langle h(\mathsf{t}_1), h(\mathsf{t}_2), \ldots, h(\mathsf{t}_n)\rangle \in h(\mathsf{R})$.

(ii) g satisfies a formula $\neg\mathsf{P}$ in $\mathfrak{M}$ if and only if g does not satisfy P in $\mathfrak{M}$.

(iii) g satisfies a formula $(\mathsf{P} \rightarrow \mathsf{Q})$ in $\mathfrak{M}$ if and only if either g does not satisfy P or g satisfies Q.

(iv) g satisfies a formula $\forall\mathsf{v}\mathsf{P}$ if and only if for all $o \in \mathsf{D}$, $g[\mathsf{v}/o]$ satisfies P.

(v) g satisfies a formula $\forall\mathsf{f}^n\mathsf{P}$ if and only if for all functions of n arguments *fun* with domain and range in D, $g[\mathsf{f}^n/\mathit{fun}]$ satisfies P.

(vi) g satisfies a formula $\forall\mathsf{F}^n\mathsf{P}$ if and only if for all sets of n-tuples R of members of D, $g[\mathsf{F}^n/R]$ satisfies P.

Existential quantification is understood accordingly. For instance, g satisfies a formula $\exists\mathsf{F}^n\mathsf{P}$ if and only if for *some* set of n-tuples R of members of D, $g[\mathsf{F}^n/R]$ satisfies P.

We define truth, thus: A *formula* P of $\mathcal{L}^2$ is *true* in a structure $\mathfrak{M}$ if and only if *every* variable assignment in $\mathfrak{M}$ satisfies P. (Notice that this allows a formula that is not a sentence to be true.)

We define logical validity in the usual way: An argument in $\mathcal{L}^2$ is *second-order valid* if and only if for every structure $\mathfrak{M}$ for $\mathcal{L}^2$ and every variable assignment g in $\mathfrak{M}$, if g satisfies every premise in $\mathfrak{M}$, then g satisfies the conclusion in $\mathfrak{M}$ as well. If an argument in $\mathcal{L}^2$ having premises A and conclusion Q is second-order valid, we write $A \models_2 \mathsf{Q}$.

Similarly, we say that a set of sentences A of $\mathcal{L}^2$ is *second-order satisfiable* if and only if there is a structure $\mathfrak{M}$ and a variable assignment g in $\mathfrak{M}$ that satisfies every member of A in $\mathfrak{M}$.

Noncompactness

Recall what we mean by *compactness*. We said (in Lesson 4) that first-order logic is compact because for every set of sentences A if every finite subset of A is first-order satisfiable (consistent), then A itself is first-order satisfiable. But we find that a similar claim cannot be made about second-order logic. (Here I follow the presentation in Enderton, p. 271.)

Observe, to begin, that for every $n \geq 2$, we have a first-order sentence λ_n which says "There are at least n things". For example, λ_3 is:

$$\exists x \exists y \exists z (x \neq y \wedge x \neq z \wedge y \neq z)$$

If you think about it, you will see that the infinite set of sentences $\{\lambda_2, \lambda_3, \lambda_4, \ldots\}$ is satisfiable in all and only structures having domains that are *at least* denumerable in cardinality (size). Here is a single sentence of second-order logic that is also satisfiable in all and only structures having domains that are at least denumerable:

$$\exists F^2 [\forall u \forall v \forall w ((F^2uv \wedge F^2vw) \rightarrow F^2uw) \wedge \forall u \neg F^2uu \wedge \forall u \exists v F^2uv]$$

What this says is that there is a relation which is transitive and irreflexive, and every individual stands in that relation to some (other) individual. Here is a simpler sentence that also is satisfiable in all and only structures having domains that are at least denumerable:

$$\lambda_\infty\text{: } \exists f^1 [\forall u \forall v ((f^1(u) = f^1(v)) \rightarrow u = v) \wedge \exists w \forall z\, f^1(z) \neq w]$$

What this says that there is a function f^1 that never yields the same output for two inputs and yet there is one object that is not in its range (a first member of the series). If we negate this sentence, the result will be a sentence, $\neg\lambda_\infty$, that is true only in structures having finite domains. Consequently, the following set of sentences is *not* second-order satisfiable:

$$\Lambda = \{\neg\lambda_\infty, \lambda_2, \lambda_3, \lambda_4, \ldots\}$$

Consider any finite subset B of Λ. That subset may contain $\neg\lambda_\infty$, but there will be some largest m such that the subset contains λ_m. So every member of B will be true in a structure having a finite domain containing m or more members. So every finite subset of Λ is satisfiable, but Λ is not itself satisfiable.

So second-order logic is not compact. We cannot say that for every set A of second-order formulas, if every finite subset is satisfiable, then A itself is satisfiable. (When we discussed compactness in Lesson 4, we were thinking of satisfiability as a property of sets of sentences. Now we are thinking of it as property of sets of *formulas*.)

Recall that we had a second formulation of compactness. We could say of first-order logic that for every first-order sentence Q and every set of first-order sentences A, if $A \models Q$, then there is some finite subset B of A such that $B \models Q$.

We cannot say this of second-order logic. Let $A = \{\lambda_2, \lambda_3, \lambda_4, \ldots\}$. Obviously, $A \models_2 \lambda_\infty$, because, as I said, A and λ_∞ are both satisfiable in exactly the structures with domains that are at least denumerable. But if B is a finite subset of A, then there will be a structure having a finite domain in which every member of B is true, and λ_∞ will be false in that domain.

Categoricity

The Löwenheim-Skolem theorems of lesson 4 (downward and upward) show that there are definite limits to the extent to which a set of sentences of a first-order language can fix the size of the domains of the structures that satisfy it. In second-order logic, by contrast, we can have theories that fix the cardinality of the domains that satisfy them, even when those domains are infinite. Indeed, we can we can write theories in second order logic that are satisfiable only in infinite domains but which are such that any two structures that satisfy them are isomorphic to one another. (See Lesson 4 for the

definition of isomorphism.) When a theory has this property (all structures that satisfy it are isomorphic to one another), we say that it is *categorical*.

Let $\mathcal{L}_A^2$ be the language of second-order arithetic (just like the language of arithmetic, except that it includes second-order variables and quantifiers). Let *second-order Peano arithmetic*, PA^2, be the conjunction of the following two sentences of $\mathcal{L}_A^2$: The universal closure of the conjunction of the nine axioms of Q (see Lesson 11), and the second-order induction scheme: $\forall \mathsf{F}(\mathsf{F}(0) \rightarrow (\forall \mathsf{x}(\mathsf{F}(\mathsf{x}) \rightarrow \mathsf{F}(\mathsf{x}')) \rightarrow \forall \mathsf{x}\mathsf{F}(\mathsf{x})))$. Obviously PA^2 is true on the intended intepretation, viz., the structure having as its domain the set of natural numbers and which assigns zero to 0, assigns the successor function to $'$, assigns the addition function to $+$, and assigns the multiplication function to $\bullet$. It can be proved that every structure that satisfies PA^2 is isomorphic to the intended interpretation. (For a proof, see Boolos and Jeffrey, chapter 18, "Second-order logic," or Shapiro, pp. 82-83. Actually, the proof does not even need N7-N9.)

Axiomatizability

The question to be considered next is whether second-order logic is axiomatizable. We can take this question in two versions.

First version: Can we write a finite set of axiom schemata and finite inference rules such that all and only second-order valid arguments are provable using axioms and inference rules of those kinds?

Let us be a little more precise. I will assume that you know what it means to say that a sentence has a certain form. I will assume that you know what it means to say that a subproof has a certain form. I will say that an *inference rule* is any rule that tells us that given finitely many premises having certain forms and finitely many subproofs having certain forms, we may derive a conclusion having a certain form. (An axiom scheme may be treated as the special case of an inference rule that says that a conclusion of a certain form may be derived from no premises and subproofs at all.) Further, I will assume that you know what a *proof* is, defined in terms of such inferences rules. For the definition, see Lesson 2. (That definition pertained only to our Barwise and Etchemendy inference rules, but the definition can be generalized.)

In these terms the question can be put this way: Is there a set of inference rules for $\mathcal{L}^2$ such that: for every argument in $\mathcal{L}^2$ there is a *proof* in this sense of the conclusion from the set of premises using these rules if and only if the argument is second-order valid? In

other words, can we have a set of inference rules that is sound and complete with respect to second-order validity?

In view of the noncompactness of second-order logic, the answer to this question is clearly *no*. Suppose, for a reductio, that the answer is *yes*. Then there is a set of inference rules such that for any set of sentences A of $\mathcal{L}^2$ and any sentence Q of $\mathcal{L}^2$ $A \models_2 \mathsf{Q}$ if and only if there is a proof of Q from A using only those inference rules; in symbols: $A \vdash_2 \mathsf{Q}$. We show that, contrary to what we have seen in the discussion of compactness, if $A \models_2 \mathsf{Q}$, then there is a finite subset of B of A such that $B \models_2 \mathsf{Q}$. Suppose $A \models_2 \mathsf{Q}$. By hypothesis, there are proofs for all second-order valid arguments; so $A \vdash_2 \mathsf{Q}$. But proofs are finite. So at most finitely many members of A are used in constructing a proof of Q from A. So there is a finite subset B of A such that $B \vdash_2 \mathsf{Q}$. By hypothesis, there are proofs *only* for second-order valid arguments: so $B \models_2 \mathsf{Q}$.

But what is the significance of the fact that there is a set of inference rules such that there is a proof for an argument using those inference rules if and only the argument is valid? From one point of view, the significance of that fact is only that if there is a such a decidable system of inference rules then the set of valid arguments (having finite sets of premises) can be effectively enumerated. (If there is such a system of inference rules, then we will be able to show that the set of codes of valid arguments is Σ_1.) But in principle there could be ways of enumerating the valid arguments (with finite sets of premises) other than one that rests on there being such a decidable set of inference rules. So a question of greater interest is this:

Second version: Is the set of second-order valid sentences of $\mathcal{L}_A^2$ effectively enumerable?

Suppose that Q is a sentence of $\mathcal{L}_A^2$ that is true in the intended interpretation of $\mathcal{L}_A^2$. Then Q is true in every structure for $\mathcal{L}_A^2$ that is isomorphic to the intended intepretation (see Lesson 4 for the idea of the proof). So since PA^2 is true on the intended interpretation and, as we have seen, PA^2 is categorical, $\mathsf{PA}^2 \models_2 \mathsf{Q}$. Thus, every true sentence of arithmetic is a semantic consequence of PA^2. Given this fact, we can prove that the set of second-order valid sentences of $\mathcal{L}_A^2$ is not effectively enumerable.

Suppose, for a reductio, that the second-order valid sentences of $\mathcal{L}_A^2$ can be effectively enumerated. If Q is *first-order* sentence of $\mathcal{L}_A^2$, then either Q or $\neg\mathsf{Q}$ is true on the intended interpretation. So, by what we have just stated, either $\mathsf{PA}^2 \models_2 \mathsf{Q}$ or $\mathsf{PA}^2 \models_2 \neg\mathsf{Q}$. So either $(\mathsf{PA}^2 \rightarrow \mathsf{Q})$ or $(\mathsf{PA}^2 \rightarrow \neg\mathsf{Q})$ is second-order valid. So if the set of second-order valid sentences of $\mathcal{L}_A^2$ were effectively enumerable, either $(\mathsf{PA}^2 \rightarrow \mathsf{Q})$ or $(\mathsf{PA}^2 \rightarrow \neg\mathsf{Q})$ would eventually show up in the enumeration. If $(\mathsf{PA}^2 \rightarrow \mathsf{Q})$ shows up, then we

will know that Q is true. If $(\mathsf{PA}^2 \rightarrow \neg \mathsf{Q})$ shows up, then we will know that Q is not true. So the set of first-order truths of arithmetic would be decidable. But since the set of first-order truths of arithmetic is not arithmetic (by Tarski's undefinability theorem, which still holds), and therefore not recursive, we know, by Church's thesis, that the set of first-order truths of arithmetic is not decidable. So the set of second-order valid sentences of $\mathcal{L}_A^2$ cannot be effectively enumerated. Consequently, the set of second-order valid sentences of any other second order language at least as rich in predicate and function symbols as $\mathcal{L}_A^2$ either.

Exercise: Suppose we have a Gödel numbering for the set of expressions of the language of second-order arithmetic, $\mathcal{L}_A^2$. We can use the formulas of the language of first-order arithmetic $\mathcal{L}_A$ to express sets of these Gödel numbers. We have also learned (though we did not prove it) that every member of $\mathcal{N}$ is a second-order consequence of PA^2. Use these facts to show that the set of codes of second-order valid formulas of $\mathcal{L}_A^2$ is not arithmetic. Hints: Modify the proof of the undecidability of first-order logic, Lesson 11, as needed. Notice that PA^2 is a particular sentence of $\mathcal{L}_A^2$ and so has a particular code. Recall that the set of codes of first-order truths of arithmetic is not arith*met*ic (by Tarski's Undefinability Theorem).

Lesson 14: Propositional Modal Logic

Modal logic is the study of the logic of necessity and possibility. The symbol that means "It is necessary that…" is $\Box$ (the box), and the symbol that means "It is possible that…" is $\Diamond$ (the diamond). These two symbols will be called *modal operators* (although strictly speaking we should call them modal *connectives*). We begin with *propositional* modal logic, which is the study of languages containing modal operators and other sentential connectives but no quantifiers.

First, we will define a *language* of propositional modal logic. Then we will define the conditions under which a sentence of the language is *true* in a modal structure. Finally, we will define several different concepts of logical validity. What we understand a modal operator to mean will depend on which definition of logical validity we accept.

The Grammar

Let the atomic sentences of $\mathcal{PM}$ be A, B, C, … Say that P is a *sentence* of $\mathcal{PM}$ if and only if either (i) P is an *atomic* sentence of $\mathcal{PM}$, or (ii) $\mathsf{P} = \neg\mathsf{Q}$ and Q is a sentence of $\mathcal{PM}$, or (iii) $\mathsf{P} = (\mathsf{Q} \rightarrow \mathsf{R})$ and both Q and R are sentences of $\mathcal{PM}$, or (iv) $\mathsf{P} = \Box\mathsf{Q}$ and Q is a sentence of $\mathcal{PM}$.

We will treat $\Diamond\mathsf{P}$ as an abbreviation of $\neg\Box\neg\mathsf{P}$.

Truth Conditions

A *frame* (or *Kripke-frame*) is a pair $\langle W, R\rangle$, where W is a nonempty set of *possible worlds*, and R is a binary relation on members of W (i.e., a set of pairs of members of W).

A *valuation V in a frame* $\langle W, R\rangle$ is a function that takes each pair consisting of an atomic sentence of $\mathcal{PM}$ and a world in W as arguments and yields either the truth value T or the truth value F as output.

For example, we might have a structure where $W = \{w_1, w_2, w_3\}$, $R = \{\langle w_1, w_1\rangle, \langle w_1, w_2\rangle, \langle w_2, w_3\rangle, \langle w_3, w_1\rangle\}$, and $V(\mathsf{A}, w_1) = \mathsf{T}$, $V(\mathsf{A}, w_2) = \mathsf{F}$, $V(\mathsf{A}, w_3) = \mathsf{T}$,….

R is called an *accessibility relation*, and if $\langle w_i, w_j\rangle \in R$, then we say that w_j is *accessible from* w_i. We abbreviate $\langle w_i, w_j\rangle \in R$ thus: w_iRw_j.

For emphasis: If w_iRw_j, then w_j is *accessible from* w_i.

We define the conditions under which sentences of $\mathcal{PM}$ are true at a world w on a valuation V in a frame $\langle W, R\rangle$ as follows:

(i) If P is atomic, then P is true at $w \in W$ on V in $\langle W, R\rangle$ if and only if $V(\mathsf{P}, w) = \mathsf{T}$.
(ii) If $\mathsf{P} = \neg\mathsf{Q}$, then P is true at $w \in W$ on V in $\langle W, R\rangle$ if and only if Q is not true at $w \in W$ on V in $\langle W, R\rangle$.
(iii) If $\mathsf{P} = (\mathsf{Q} \rightarrow \mathsf{R})$, then P is true at $w \in W$ on V in $\langle W, R\rangle$ if and only if either Q is not true at $w \in W$ on V in $\langle W, R\rangle$ or R is true at $w \in W$ on V in $\langle W, R\rangle$.
(iv) If $\mathsf{P} = \Box\mathsf{Q}$, then P is true at $w \in W$ on V in $\langle W, R\rangle$ if and only if for *all* $w' \in W$, if wRw', then Q is true at $w' \in W$ on V in $\langle W, R\rangle$.

A consequence is that:
(v) If $\mathsf{P} = \Diamond\mathsf{Q}$, then P is true at w on V in $\langle W, R\rangle$ if and only if for *some* $w' \in W$, wRw', and Q is true at w' on V in $\langle W, R\rangle$.

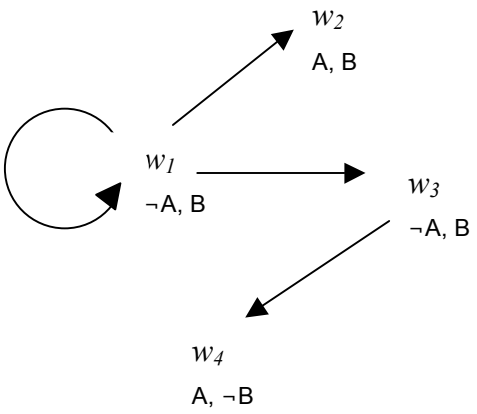
Proof: $\Diamond\mathsf{Q}$ abbreviates $\neg\Box\neg\mathsf{Q}$, and, by (2) and (4), $\neg\Box\neg\mathsf{Q}$ is true at w on V in $\langle W, R\rangle$ if and only if for *some* $w' \in W$, wRw', and Q is true at w' on V in $\langle W, R\rangle$.

A sentence is *false* at a world on a valuation in a frame if and only if it is not *true* at that world on that valuation in that frame.

If we suppose that one of the worlds in W is the actual world (the world that really exists), then we can say that A is *true* (*simpliciter*) if and only if A is true at the *actual* world.

(From now on, I will just assume that the worlds we're talking about are in W in the frame we're talking about.)

For example:



(Next page for explanation)

$W = \{w_1, w_2, w_3, w_4\}$, and R is represented by the arrows. $\Diamond$A is true at w_1 in this frame, because A is true at w_2 in this frame, and w_1Rw_2. $\Box$A is false at w_1 in this frame, because A is false at w_3 in this frame, and w_1Rw_3. However, $\Box$B is true at w_1 in this frame, because B is true at every world $w \in W$ such that w_1Rw. (B is false at w_4, but it does not matter, since w_4 is not accessible from w_1. We are not assuming – at this point – that accessibility is transitive.)

Kinds of Frame

Towards defining different kinds of validity, we define different kinds of frame:

A frame $\langle W, R\rangle$ is *reflexive* if and only if for all $w \in W$, wRw. (Every world is accessible from itself.)

A frame $\langle W, R\rangle$ is *symmetric* if and only if for all $w, w' \in W$, if wRw' then $w'Rw$. (If w' is accessible from w, then w is accessible from w'.)

A frame $\langle W, R\rangle$ is *transitive* if and only if for all $w, w', w'' \in W$, if wRw' and $w'Rw''$, then wRw''. (If w'' is accessible from w' and w' is accessible from w, then w'' is accessible from w.)

A frame is a *T*-frame if and only if it is reflexive.

A frame is a *B*-frame if and only if it is reflexive and symmetric.

A frame is an *S4*-frame if and ony if it is reflexive and transitive.

A frame is an *S5*-frame if and only if it is reflexive, symmetric and transitive.

Note: In an *S5*-frame, W can be divided into mutually exclusive cells such that W is the union of all of the cells and such that for each cell $C \subseteq W$, for all $w, w' \in C$, wRw'. In other words, if $w_1, w_2 \in W$ and w_1Rw_2, then for all w_3 such that w_1Rw_3 and for all w_4 such that w_2Rw_4, w_3Rw_4, and if $w_1, w_2 \in W$ and $\neg w_1Rw_2$, then for all w_3 such that w_1Rw_3 and for all w_4 such that w_2Rw_4, $\neg w_3Rw_4$.

Kinds of Validity

Let A be a set of sentences of $\mathcal{PM}$ and Q be a sentence of $\mathcal{PM}$.

The argument having the sentences in A as premises and the sentence Q as its conclusion is K-valid ($A \models_K \mathsf{Q}$) if and only if for *every* frame $\langle W, R\rangle$ and every valuation V in $\langle W, R\rangle$ and every world $w \in W$, if every member of A is true at w on V in $\langle W, R\rangle$, then Q is true at w on V in $\langle W, R\rangle$.

The argument having the sentences in A as premises and the sentence Q as its conclusion is T-valid ($A \models_T \mathsf{Q}$) if and only if for every T-frame $\langle W, R\rangle$ (reflexive) and every valuation V in $\langle W, R\rangle$ and every world $w \in W$, if every member of A is true at w on V in $\langle W, R\rangle$, then Q is true at w on V in $\langle W, R\rangle$.

The argument having the sentences in A as premises and the sentence Q as its conclusion is B-valid ($A \models_B \mathsf{Q}$) if and only if for every B-frame $\langle W, R\rangle$ (reflexive and symmetric) and every valuation V in $\langle W, R\rangle$ and every world $w \in W$, if every member of A is true at w on V in $\langle W, R\rangle$, then Q is true at w on V in $\langle W, R\rangle$.

The argument having the sentences in A as premises and the sentence Q as its conclusion is $S4$-valid ($A \models_{S4} \mathsf{Q}$) if and only if for every $S4$-frame $\langle W, R\rangle$ (reflexive and transitive) and every valuation V in $\langle W, R\rangle$ and every world $w \in W$, if every member of A is true at w on V in $\langle W, R\rangle$, then Q is true at w on V in $\langle W, R\rangle$.

The argument having the sentences in A as premises and the sentence Q as its conclusion is $S5$-valid ($A \models_{S5} \mathsf{Q}$) if and only if for every $S5$-frame $\langle W, R\rangle$ (reflexive and symmetric and transitive) and every valuation V in $\langle W, R\rangle$ and every world $w \in W$, if every member of A is true at w on V in $\langle W, R\rangle$, then Q is true at w on V in $\langle W, R\rangle$.

Proof theory

Corresponding to each of these validity concepts, we specify a set of axioms such that all and only the arguments that are valid in that sense are provable using those axioms and Modus Ponens.

At the core of each set of axioms are the axioms of *PL* from Lesson 5, which I repeat here:

The axiom system PL (for $\mathcal{PM}$):

L1: $(P \rightarrow (Q \rightarrow P))$
L2: $(P \rightarrow (Q \rightarrow R)) \rightarrow ((P \rightarrow Q) \rightarrow (P \rightarrow R))$
L3: $((\neg P \rightarrow \neg Q) \rightarrow (Q \rightarrow P))$

Every sentence of $\mathcal{PM}$ that has one of these three forms will be an axiom of *PL* (for $\mathcal{PM}$). We could just as well say: Every tt-valid sentence of $\mathcal{PM}$ is an axiom, because for any language tt-validity is decidable. In any case, we will assume that every tt-valid sentence can be derived from axioms of these three forms by means of repeated applications of Modus Ponens.

The System K

The system *K* adds to the axioms of *PL* one axiom scheme and one inference rule. The axiom scheme is:

L4: $(\Box(P \rightarrow Q) \rightarrow (\Box P \rightarrow \Box Q))$

This is called the *characteristic axiom scheme* of *K*. The additional inference rule is:

Necessitation: If P can be derived from axioms only (no special premises), then infer $\Box P$.

When there exists a proof of Q using only sentences in set *A* and axioms of any of the forms, *L1-L4*, Modus Ponens and Necessitation, we write $A \vdash_K Q$.

We say that *K* = *PL* + *L4* + *Necessitation*.

As a matter of fact, $A \vdash_K Q$ if and only if $A \models_K Q$, but we won't prove that. In other words, proof in *K* is sound and complete with respect to *K*-validity.

Exercise 1: Show that $\Box(P \rightarrow Q) \models_K (\Box P \rightarrow \Box Q)$. (Hint: Suppose that the premise is true at some world on some valuation in some *K*-frame and that the conclusion is false at that world on that valuation in that *K*-frame, and then derive a contradiction.)

Exercise 2: Show that $\Box P \not\models_K P$. (So the box does not yet behave very much like a necessity operator.)

Solution: Let $W = \{w_1, w_2\}$. Suppose that $R = \{\langle w_1, w_2\rangle\}$. (Note that R is not reflexive.) Suppose $V(\mathsf{A}, w_1) = \mathsf{F}$, but $V(\mathsf{A}, w_2) = \mathsf{T}$. Since for every $w \in W$ such that w_1Rw (namely, w_2 only) A is true at w, $\Box\mathsf{A}$ is true at w_1. But A is not true at w_1.

The System T

The system *T* adds the following axiom scheme:

L5: $(\Box\mathsf{P} \rightarrow \mathsf{P})$

This is the *characteristic axiom scheme* of *T*. So $T = K + L5$.

As a matter of fact, $A \vdash_T \mathsf{Q}$ if and only if $A \models_T \mathsf{Q}$, but we won't prove that.

Exercise 3: Prove that $\Box\mathsf{P} \models_T \mathsf{P}$.

Solution: Let $\langle W, R\rangle$ be a *T*-frame, and suppose that V is a valuation such that $\Box\mathsf{P}$ is true at w on V in $\langle W, R\rangle$. Then, by condition (iv) in the definition of truth, for every $w' \in W$ such that wRw', P is true at w' on V in $\langle W, R\rangle$. But since $\langle W, R\rangle$ is reflexive in every *T*-frame, wRw. So P is true at w on V in $\langle W, R\rangle$.

Exercise 4: Prove that $\mathsf{P} \not\models_T \Box\Diamond\mathsf{P}$.

Solution: Suppose $W = \{w_1, w_2\}$ and $R = \{\langle w_1, w_1\rangle, \langle w_2, w_2\rangle, \langle w_1, w_2\rangle\}$. $\langle W, R\rangle$ is a *T*-frame since R is reflexive. (But note that R is not symmetric.) Suppose $V(\mathsf{A}, w_1) = \mathsf{T}$ and $V(\mathsf{A}, w_2) = \mathsf{F}$. A is true at w_1 on V in $\langle W, R\rangle$. But the only world accessible from w_2 is w_2 itself, and A is false at w_2. So $\Diamond\mathsf{A}$ is false at w_2 on V in $\langle W, R\rangle$. But w_1Rw_2; so we cannot say that for all $w \in W$, if w_1Rw, then $\Diamond\mathsf{A}$ is true on V in $\langle W, R\rangle$. So $\Box\Diamond\mathsf{A}$ is not true at w_1 on V in $\langle W, R\rangle$.

Exercise 5: Prove that $\Box\mathsf{P} \not\models_T \Box\Box\mathsf{P}$.

The System B

The system *B* adds to the system *T* the following axiom scheme:

L6: $(\mathsf{P} \rightarrow \Box\Diamond\mathsf{P})$

This is the *characteristic axiom scheme* of *B*. So $B = T + L6$.

As a matter of fact, $A \vdash_B \mathsf{Q}$ if and only if $A \models_B \mathsf{Q}$, but we won't prove that.

Exercise 6: Prove that $\mathsf{P} \models_B \Box\Diamond\mathsf{P}$.

Solution: Suppose that P is true at w on V in a *B*-frame $\langle W, R\rangle$. Consider an arbitrary world $w' \in W$ such that wRw'. Since $\langle W, R\rangle$ is symmetric, $w'Rw$. So, since P is true at w, $\Diamond\mathsf{P}$ is true at w'. So for every $w' \in W$ such that wRw', $\Diamond\mathsf{P}$ is true at w'. So $\Box\Diamond\mathsf{P}$ is true at w.

Exercise 7: Prove that $\Box\mathsf{P} \not\models_B \Box\Box\mathsf{P}$.

Solution: Let $W = \{w_1, w_2, w_3\}$ and $R = \{\langle w_1, w_1\rangle, \langle w_2, w_2\rangle, \langle w_3, w_3\rangle, \langle w_1, w_2\rangle, \langle w_2, w_1\rangle, \langle w_2, w_3\rangle, \langle w_3, w_2\rangle\}$. So $\langle W, R\rangle$ is a *B*-frame, since it is reflexive and symmetric. (But note that $\langle W, R\rangle$ is not transitive.) Suppose $V(\mathsf{A}, w_1) = \mathsf{T}$, and $V(\mathsf{A}, w_2) = \mathsf{T}$, but $V(\mathsf{A}, w_3) = \mathsf{F}$. Since only w_1 and w_2 are accessible from w_1, $\Box\mathsf{A}$ is true at w_1. But since w_2Rw_3 and A is not true at w_3, $\Box\mathsf{A}$ is not true at w_2. So since w_1Rw_2 and $\Box\mathsf{A}$ is not true at w_2, $\Box\Box\mathsf{A}$ is not true at w_1.

The System S4

The system *S4* adds to the system *T* the following axiom scheme:

L7: $(\Box\mathsf{P} \rightarrow \Box\Box\mathsf{P})$

This is the *characteristic axiom scheme* of *S4*. So $S4 = T + L7$. Notice that *S4* builds on *T*, not *B*, and does not include *L6*.

As a matter of fact, $A \vdash_{S4} \mathsf{Q}$ if and only if $A \models_{S4} \mathsf{Q}$, but we won't prove that.

Exercise 8: Prove that $\Box\mathsf{P} \models_{S4} \Box\Box\mathsf{P}$.

Solution: Suppose $\Box\Box\mathsf{P}$ is false at w on V in an *S4*-frame $\langle W, R\rangle$. So there is a world $w' \in W$ such that wRw' and $\Box\mathsf{P}$ is false at w'. So there is a world w'' such that $w'Rw''$ such that P is false at w''. But $\langle W, R\rangle$ is transitive. So, since wRw' and $w'Rw''$, wRw''. So $\Box\mathsf{P}$ is false at w.

Exercise 9: Prove that $P \not\models_{S4} \Box\Diamond P$.

Solution: See the proof that $P \not\models_{T} \Box\Diamond P$.

The System S5

The system *S5* adds to the system *T* both the characteristic axiom of *B* and the characteristic axiom of *S4*:

L6: $(P \rightarrow \Box\Diamond P)$
L7: $(\Box P \rightarrow \Box\Box P)$

So *S5* = *T* + *L6* + *L7* = *B* + *L7* = *S4* + *L6*.

As a matter of fact, $A \vdash_{S5} Q$ if and only if $A \models_{S5} Q$, but we won't prove that.

The Reducibility of Modalities in S5

Let $P \dashv\models Q$ mean: $P \models_{S5} Q$ and $Q \models_{S5} P$.

Theorem (the reducibilities of modalities in S5):

(i) $\Diamond P \dashv\models \Box\Diamond P$
(ii) $\Box P \dashv\models \Diamond\Box P$
(iii) $\Diamond P \dashv\models \Diamond\Diamond P$
(iv) $\Box P \dashv\models \Box\Box P$

Before we prove this, let us contemplate what it means. It means that whenever we have a sentence that begins with a string of modal operators, we can find an equivalent formula that begins with just the last of those operators. For example:

$\Box\Diamond\Diamond\Box P \dashv\models \Diamond\Diamond\Box P \dashv\models \Diamond\Box P \dashv\models \Box P$

Now the *proof:*

(i) R-L: By exercise 3, $\Box Q \models_{S5} Q$. So $\Box\Diamond P \models_{S5} \Diamond P$.
L-R: Suppose that $\Diamond P$ is true at *w*. Then there is a world *w′* accessible from *w* where P is true. But *w′* is accessible from every world accessible from *w*. So at every world accessible from *w*, $\Diamond P$ is true. So $\Box\Diamond P$ is true at *w*.

(ii) From (i) we have $\neg\Diamond\neg Q \dashv\models \neg\Box\Diamond\neg Q$. Applying the definition of $\Diamond$, this gives us $\Box Q \dashv\models \Diamond\Box Q$.

(iii) R-L: Suppose $\Diamond\Diamond P$ is true at *w*. So $\Diamond P$ is true at some world *w′* accessible from *w*. So P is accessible from some world *w″* accessible from *w′*. But *w″* is accessible from *w*. So $\Diamond P$ is true at *w*.
L-R: By exercise 3, $\Box\neg Q \models_{S5} \neg Q$. So $\neg\neg Q \models_{S5} \neg\Box\neg Q$. So $Q \models_{S5} \Diamond Q$. So $\Diamond P \models_{S5} \Diamond\Diamond P$.

(iv) From (iii) we have $\neg\Diamond\neg Q \dashv\models \neg\Diamond\Diamond\neg Q$. Applying the definition of $\Diamond$, this gives us $\Box P \dashv\models \Box\Box P$.

Some Objections

The contemporary literature on propositional modal logic recognizes nothing the least bit controversial. (As we will see, that is not the case when it comes to quantified modal logic.) Although I may be the only one, I think that actually *none* of these logics captures the logic of natural language modal operators. That is because what we really need is a three-valued semantics.

First, let's compare a two-valued semantics to a three-valued semantics with respect to a couple of points:

Inconsistency and implication: In a two-valued semantics, if P and Q are inconsistent, then P implies $\neg Q$. If there is no valuation on which both P and Q are true, then for every valuation on which P is true $\neg Q$ is true too. But in a three-value semantics, this does not hold. There might be no valuation on which P and Q are both true, and yet there may be a valuation on which P is true and $\neg Q$ is neither true nor false.

Contraposition: A two-valued semantics obeys the law of contraposition: If P implies Q, then $\neg Q$ implies $\neg P$. (We made tacit use of this in the proof of the reducibility of modalities in *S5*.) But a three-value semantics need not obey this law. Suppose P implies Q, because on every interpretation on which P is true, Q is true. Then on every interpretation on which Q is *not* true, P is *not* true. But every interpretation on which

$\neg Q$ is true is an interpretation on which Q is not true. So every interpretation on which $\neg Q$ is true is an intepretation on which P is not true. But it does not follow that every interpetation on which $\neg Q$ is true is an interpretation on which $\neg P$ is true, because P could fail to be true by being neither true nor false, in which case $\neg P$ is neither true nor false as well.

First problem case:

Suppose I step outside and declare, "It's going to rain!" But then I look up at the sky, reconsider and say, "Well, it might not". Doesn't this count as *taking back* what I said in the first place? If so, there there is a kind of inconsistency between P and $\Diamond\neg P$. But in a two-valued logic, if P and Q are inconsistent, then P implies $\neg Q$. We don't want to say that P implies $\neg\Diamond\neg P$, because that means that P implies $\Box P$, which is certainly not the case. So to maintain that P and $\Diamond\neg P$ are inconsistent, we have to go to a three-valued semantics.

Second problem case:

If it is *possible* that it is *possible* that I will not go for a walk, then surely it is *possible* that I will not go for a walk. But suppose that is *necessary* that I will go for a walk. May I infer that it is *necessary* that it be *necessary* that I will go for a walk? No! The reason why it is necessary that I go for walk may be that I *decided* to go for a walk, and I always do what I have decided to do. But it is not necessary that it be necessary that I go for a walk, because I did not have to decide to go for a walk. So $\Diamond\Diamond\neg P$ implies $\Diamond\neg P$, but $\Box P$ does not imply $\Box\Box P$. But since the principle of contraposition holds in a two-valued semantics, the only way we can have the one implication without the other is by going to a three-valued semantics.

Third problem case:

I will be in Cincinnati on Friday. Therefore, it is *necessarily possible* that I will be in Cincinnati on Friday. But it *may* be *necessary* that I *not* be in Cincinnati on Monday. Does it follow that I will not be in Cincinnati on Monday? I think not. In other words, P implies $\Box\Diamond P$, but $\Diamond\Box\neg P$ does not imply $\neg P$. But here too, since the principle of contraposition holds in a two-valued semantics, the only way we can have the one implication without the other is by going to a three-valued semantics.

Lesson 15: Quantified Modal Logic

Here's our agenda: First we will look at the "obvious" way of interpreting sentences containing both quantifiers and modal operators. Then we will look at some reasons to be dissatisfied with that. Then we will investigate two alternatives (the "free-logical" alternative, and the "existence predicate" alternative). Finally, we will draw the distinction between rigid and non-rigid designation. We will concern ourselves exclusively with questions of semantics and will not deal with proof theory (axiom systems) at all.

Throughout we will suppose that we have a language $\mathcal{QM}$, which, with respect to grammar, is just like a first-order language except that it contains the box, $\Box$, which behaves, grammatically just like the negation sign. We will dispense with function symbols for present purposes.

Simple Quantified Modal Logic

In Simple Quantified Modal Logic (a term I just made up), we simply add to each frame a set of objects, the domain. So a frame is now a triple, $\langle D, W, R\rangle$, where D is a nonempty set of objects, and, as before, W is a nonempty set of worlds and R is an accessibility relation on the worlds in W. For simplicity I will assume that for every frame $\langle D, W, R\rangle$, $\langle W, R\rangle$ is an *S5*-frame (and therefore, reflexive, symmetic and transitive). (This means that we could drop all mention of R and just assume that all worlds in W were accessible to one another.)

We will define a *modal structure* as a quadruple $\langle \Sigma, D, W, R\rangle$, where $\langle D, W, R\rangle$ is a frame, as just defined, and Σ is an *interpretation*. An interpretation Σ is a function of two arguments. The arguments are either a world in W and an individual constant, or a member of W and a predicate (i.e., predicate constant). For each $w \in W$ and each individual constant n of $\mathcal{QM}$, $\Sigma(\mathsf{n}, w) \in D$, and for each $w \in W$ and each n-ary predicate F of $\mathcal{QM}$, $\Sigma(\mathsf{F}, w)$ = a set of n-tuples of members of D. $\Sigma(\mathsf{a}, w)$ is the *denotation* of a at w, and $\Sigma(\mathsf{F}, w)$ is the *extension* of F at w.

Furthermore, we *stipulate* that for each individual constant n and for all $w, w' \in W$, $\Sigma(\mathsf{n}, w) = \Sigma(\mathsf{n}, w')$. (So it might have been simpler to have two kinds of interpretation, one for individual constants, which did *not* take worlds as arguments at all, and one for predicates which *did* take worlds as arguments.) This is what it means to say that an individual constant is a *rigid designator*: It denotes the same object at every world.

Note an ambiguity in the phrase "denotes at a world". What we really mean here is *denotes relative to* a world. There is a possible world in which "Barack Obama" is the name of Sarah Palin. We could say that *in* that world "Barack Obama" denotes Sarah Palin. But that's not the kind of thing we mean here when we speak of a name as denoting something at a world. In English, as it is spoken in the actual world, "Barack Obama" denotes Barack Obama, not Sarah Palin, relative to, or "at", every world.

Similarly, we relativize *variable assignments* (in structures) to worlds. A function g is a variable assignment in a structure $\langle \Sigma, D, W, R \rangle$ if and only if for each variable v and world $w \in W$, $g(\mathsf{v}, w) \in D$. We stipulate that for all $w, w' \in W$, $g(\mathsf{v}, w) = g(\mathsf{v}, w')$ The variant $g[\mathsf{v}/o]$ of variable assignment g is the variable assignment just like g except that for each $w \in W$, $g[\mathsf{v}/o](\mathsf{v}, w) = o$.

Term assignments are defined in terms of variable assignments and interpretations. Where t is a term and g is some variable assignment in $\langle \Sigma, D, W, R \rangle$,

$$h(\mathsf{t}, w) = \begin{cases} \Sigma(\mathsf{t}, w) \text{ if } \mathsf{t} \text{ is an individual constant.} \\ g(\mathsf{t}, w) \text{ if } \mathsf{t} \text{ is a variable.} \end{cases}$$

h is a *term assignment* for $\langle \Sigma, D, W, R \rangle$ and g.

Next, we define satisfaction of a formula by a variable assignment in a modal structure:

For every formula P and every structure $\langle \Sigma, D, W, R \rangle$ and every world $w \in W$, and every variable assignment g in $\langle \Sigma, D, W, R \rangle$, g *satisfies* P in $\langle \Sigma, D, W, R \rangle$ at w if and only if:

(A) $\mathsf{P} = \mathsf{R}(\mathsf{t}_1, \mathsf{t}_2, \ldots, \mathsf{t}_n)$, where R is an n-ary predicate and $\mathsf{t}_1, \mathsf{t}_2, \ldots, \mathsf{t}_n$ are n terms, and $\langle h(\mathsf{t}_1, w), h(\mathsf{t}_2, w), \ldots, h(\mathsf{t}_n, w) \rangle \in \Sigma(\mathsf{R}, w)$, or,

(¬) $\mathsf{P} = \neg\mathsf{Q}$ and g does not satisfy Q in $\langle \Sigma, D, W, R \rangle$ at w, or

(→) $\mathsf{P} = (\mathsf{Q} \rightarrow \mathsf{R})$ and either g does not satisfy Q in $\langle \Sigma, D, W, R \rangle$ at w or g satisfies R in $\langle \Sigma, D, W, R \rangle$ at w, or

(∀) $\mathsf{P} = \forall\mathsf{v}\mathsf{Q}$ and for all $o \in D$, $g[\mathsf{v}/o]$ satisfies Q in $\langle \Sigma, D, W, R \rangle$ at w, or

(□) $\mathsf{P} = \Box\mathsf{Q}$ and for all $w' \in W$, if wRw' then g satisfies Q in $\langle \Sigma, D, W, R \rangle$ at w'.

We say that a sentence P of $\mathcal{QM}$ is *true* in $\langle \Sigma, D, W, R \rangle$ at w if and only if for every variable assignment g in $\langle \Sigma, D, W, R \rangle$ at w, g satisfies P in $\langle \Sigma, D, W, R \rangle$ at w.

If A is a set of sentences of $\mathcal{QM}$ and Q is a sentence of $\mathcal{QM}$, then $A \models_{\text{SQML}} Q$ if and only if for every modal structure $\langle \Sigma, D, W, R \rangle$ and every $w \in W$, if every member of A is true in $\langle \Sigma, D, W, R \rangle$ at w, then Q is true in $\langle \Sigma, D, W, R \rangle$ at w.

The Barcan and Converse Barcan Formulas:

The problem with this shotgun wedding of quantifier domains and relativization of truth to worlds is that the Barcan and converse Barcan formulas turn out to be valid. (The validity of the Barcan formula was first discussed in the 1940's by Ruth Barcan Marcus.)

The Barcan formula:
$\models_{\text{SQML}} (\forall v \Box Q \rightarrow \Box \forall v Q)$

The Barcan formula (existential variant):
$\models_{\text{SQML}} (\Diamond \exists v Q \rightarrow \exists v \Diamond Q)$

The converse Barcan formula:
$\models_{\text{SQML}} (\Box \forall v Q \rightarrow \forall v \Box Q)$

The converse Barcan formula (existential variant):
$\models_{\text{SQML}} (\exists v \Diamond Q \rightarrow \Diamond \exists v Q)$

Proof of the Barcan Formula: Suppose $\forall v \Box Q$ is true at w. Then for every variable assignment g and every object $o \in D$, $g[v/o]$ satisfies $\Box Q$ at w. So for every variable assignment g and every object $o \in D$, for every world $w' \in W$ such that wRw', $g[v/o]$ satisfies Q at w'. So for every variable assignment g, for every $w' \in W$ such that wRw', for every object $o \in D$, $g[v/o]$ satisfies Q at w'. So for every variable assignment g, for all $w' \in W$ such that wRw', g satisfies $\forall v Q$ at w'. So for every variable assignment g, g satisfies $\Box \forall v Q$ at w. So $\Box \forall v Q$ is true at w.

Proof of the Converse Barcan Formula: Similar.

The essential point in both of these proofs is that the domain of objects that we have to "look at" in evaluating a quantified formula is entirely independent of the domain of worlds that we have to look at in evaluating a boxed formula. So we can reverse the

order of our metalinguistic quantifiers in formulating the satisfaction conditions of a universally quantified boxed formula or a boxed universally quantified formula.

This does not mean that the order of *any* series of modal operators and quantifiers can be switched around. For example, we have:

$\not\models_{\text{SQML}} (\Box\exists v Q \rightarrow \exists v \Box Q)$

Proof: Suppose $W = \{w, w'\}$, $D = \{a, b\}$, $\Sigma(F, w) = \{\langle a \rangle\}$, $\Sigma(F, w') = \{\langle b \rangle\}$. Let g be some variable assignment in this structure. $g[x/a]$ satisfies Fx at w. So g satisfies $\exists x Fx$ at w. $g[x/b]$ satisfies Fx at w'. So g satisfies $\exists x Fx$ at w'. So g satisfies $\Box\exists x Fx$ at w. But $g[x/a]$ does not satisfy Fx at w'; so $g[x/a]$ does not satisfy $\Box Fx$ at w. And $g[x/b]$ does not satisfy Fx at w; so $g[x/b]$ does not satisfy $\Box Fx$ at w. So g does not satisfy $\exists x \Box Fx$ at w. So g does not satisfy $(\Box\exists x Fx \rightarrow \exists x \Box Fx)$ at w. So $(\Box\exists x Fx \rightarrow \exists x \Box Fx)$ is not true at w.

To see that this is a right result consider the following conditional: "If necessarily there is some number that is the number of planets, then there is some number that is necessarily the number of planets." Or: "If necessarily something exists, then there is something that necessarily exists".

Counterexamples to the Barcan formula: Suppose that everything that actually exists has (rest) mass. It is plausible that each of those things necessarily has mass, because for anything that has mass in fact, it just would not be recognizable as the same thing if it did not have mass. So everything necessarily has mass. But even in that case there there could be a possible world in which massless things exist. So it is not the case that necessarily everything has mass.

Or consider the Barcan formula in its existential variant. I do not have a twin brother. But my having a twin brother is a possibility. That is, there is a possible world in which I have a twin brother. So "It is possible that I have a twin brother" is true. But it does not follow that there is someone now, someone out there, maybe somewhere in France, of whom we can say: It is possible that *he* is my twin brother. So, "There is someone who is possibly my twin brother" is false. (Admittedly, the premise sounds a little odd, at least if we assume that I know that I do not have a twin brother. But suppose I don't know this. Perhaps I suspect that I might have a twin brother because once, when I was very young, I heard my mother and father saying some things that I did not exactly understand.)

Counterexamples to the Converse Barcan Formula: In each possible world, it is true that everything is something: $\forall x \exists y\ x = y$. Since this is true at each world, we seem to have it that the following is true at our world: $\Box\forall x \exists y\ x = y$. But now look at *that couch*. That couch does not exist in every possible world, does it? I hope not! So we cannot say that in each possible world *it* is something. So the following sentence is false: $\forall x \Box \exists y\ x = y$.

Modal Free Logic

Some people say that what has gone wrong is that in SQML we have assumed that the domain of objects relative to which we evaluate quantified sentences is the same for every world. In other words, we have assumed that for each world the objects "in" that world are the same as the objects "in" every other world. What I am calling Modal Free Logic (MFL) remedies that purported error. The term "Free-logic" actually refers to a kind of semantics and matching proof theory in which we allow individual constants that do not denote anything. (The main exponents of this have been Karel Lambert and Ermanno Bencivenga.) We will not allow denotationless terms, but we have to modify Universal Elimination and Existential Introduction in ways that are similar to the way they are modified in Free Logic; hence the name.

In Modal Free Logic, a structure is a quintuple $\langle \Sigma, \delta, D, W, R \rangle$, where Σ, D, W, and R are as before. δ is a function on members of W that yields, for each member of W, a subset of D. So for each $w \in W$, $\delta(w) \subseteq D$. We think of $\delta(w)$ as the set of things that "exist" at w. We call D the *outer domain* (of the structure $\langle \Sigma, \delta, D, W, R \rangle$) and we call $\delta(w)$ the *inner domain* for w (in $\langle \Sigma, \delta, D, W, R \rangle$). So Chris Gauker is in the inner domain for the actual world, and Sherlock Holmes may be in the outer domain, but he is not in the inner domain for the actual world.

(Or so they say. I myself don't understand what sense it makes to say that Sherlock Holmes—*that very person*—inhabits some possible world. "What person?", you may rightly ask. "There *is* no Sherlock Holmes!")

The satisfaction conditions can be written the same way as for SQML, except that we have to put "$\langle \Sigma, \delta, D, W, R \rangle$" in place of "$\langle \Sigma, D, W, R \rangle$", and in place of ($\forall$) we have:

($\forall$F) $\mathsf{P} = \forall \mathsf{v}\mathsf{Q}$ and for all $o \in \delta(w)$, $g[\mathsf{v}/o]$ satisfies Q in $\langle \Sigma, \delta, D, W, R \rangle$ at w, or …

In MFL, neither the Barcan Formula nor the Converse Barcan Formula is valid. To see that the Barcan Formula, $(\forall \mathsf{v} \Box \mathsf{Q} \rightarrow \Box \forall \mathsf{v} \mathsf{Q})$, is not valid, consider a structure in which

there is a world w such that for all $o \in \delta(w)$, $\langle o \rangle \in \Sigma(\mathsf{F}, w)$ and for all w', if wRw', then $\langle o \rangle \in \Sigma(\mathsf{F}, w')$. In other words, every object in w is F in w and is F in every world accessible from w. $\forall \mathsf{x} \Box \mathsf{Fx}$ will be true at w. But suppose also that in some worlds w', there are objects in $\delta(w')$ that are not in $\Sigma(\mathsf{F}, w')$ (and so not in $\delta(w)$). $\Box \forall \mathsf{x} \mathsf{Fx}$ will be false at w.

To see that the Converse Barcan Formula is not valid, consider a structure in which for every world $w \in W$, for all $o \in \delta(w)$, $\langle o \rangle \in \Sigma(\mathsf{F}, w)$ but in which for some $w \in W$, and some $w' \in W$, and some $o' \in \delta(w')$, $\langle o' \rangle \notin \Sigma(\mathsf{F}, w)$. (Recall that $\langle W, R \rangle$ is an *S5* frame.) $\Box \forall \mathsf{x} \mathsf{Fx}$ will be true at w', but $\forall \mathsf{x} \Box \mathsf{Fx}$ will be false at w'. In other words, at each world everything that exists at that world is F, but some things that exist at w' are not F at w.

But while MFL has these seemingly desirable results, it has has some strange results too. In particular, Existential Introduction is no longer valid: $\mathsf{Q} \not\models_{\mathrm{MFL}} \exists \mathsf{v} \mathsf{Q}\mathsf{v}/\mathsf{n}$. For example, $\mathsf{Fa} \not\models_{\mathrm{MFL}} \exists \mathsf{x} \mathsf{Fx}$. Suppose $\Sigma(\mathsf{a}, w) \in D$, and $\langle \Sigma(\mathsf{a}, w) \rangle \in \Sigma(\mathsf{F}, w)$, but for all $o \in \delta(w)$, $\langle o \rangle \notin \Sigma(\mathsf{F}, w)$. Then Fa will be true at w, but $\exists \mathsf{x} \mathsf{Fx}$ will be false at w. The most we can say is that $\{\mathsf{Q}, \exists \mathsf{v}\ \mathsf{v} = \mathsf{n}\} \models_{\mathrm{MFL}} \exists \mathsf{v} \mathsf{Q}\mathsf{v}/\mathsf{n}$. (Similarly, we have $\forall \mathsf{v} \mathsf{Q} \not\models_{\mathrm{MFL}} \mathsf{Q}\mathsf{n}/\mathsf{v}$, but $\{\forall \mathsf{v} \mathsf{Q}, \exists \mathsf{v}\ \mathsf{v} = \mathsf{n}\} \models_{\mathrm{MFL}} \mathsf{Q}\mathsf{n}/\mathsf{v}$.)

In defense of this result, it may be said that "Santa Claus lives at the North Pole" is true, while "There exists an x such that x lives at the North Pole" is false. I don't see it. "Santa Claus lives at the North Pole" is false. (That's not to say that "It is not the case that Santa Claus lives at the North Pole" is true. Both sentences may be neither true nor false. To deal with a language containing names of fictions, we may need a three-valued semantics.)

Off hand, you might think that it would help if we stipulated that each extension at a world must be formed from members of the inner domain for that world. That is, $\Sigma(\mathsf{F}, w)$ must be a set of n-tuples formed from members of $\delta(w)$, not merely members of D. Then if Fa is true, so that $\langle \Sigma(\mathsf{a}, w) \rangle \in \Sigma(\mathsf{F}, w)$, we can be sure that $\Sigma(\mathsf{a}, w) \in \delta(w)$. So $\mathsf{Fa} \models_{\mathrm{MFL}} \exists \mathsf{x} \mathsf{Fx}$. But that only helps when the premise is an atomic sentence (and in certain other cases). We still have: $\neg \mathsf{Ga} \not\models_{\mathrm{MFL}} \exists \mathsf{x} \neg \mathsf{Gx}$. If $\Sigma(\mathsf{a}, w) \notin \delta(w)$, then, by our stipulation about extensions, $\neg \mathsf{Ga}$ will be true at w; but $\exists \mathsf{x} \neg \mathsf{Gx}$ may still be false at w (and will be false if for all $o \in \delta(w)$, $\langle o \rangle \in \Sigma(\mathsf{G}, w)$).

Alternatively, we might stipulate that for every name a, $\Sigma(\mathsf{a}, w) \in \delta(w)$. That will ensure that Existential Introduction is valid without restriction. But that solution carries costs of its own. To invalidate the Barcan formula we want to allow that for some w, w' $\delta(w) \neq \delta(w')$. So if we stipulate that $\Sigma(\mathsf{a}, w) \in \delta(w)$ and $\Sigma(\mathsf{a}, w') \in \delta(w')$, then we cannot stipulate that for all w, w', $\Sigma(\mathsf{a}, w) = \Sigma(\mathsf{a}, w')$, which means that we cannot stipulate that

names are rigid designators. The undesirable consequence of that is that, without further ado, we will not be able to prove the validity of $\Box a = a$. We might restore the necessity of identity by stipulating that, for each structure, the extension of "=" is the identity relation on the entire outer domain. But then we will have to decide whether we want $a = a$ to imply $\exists x\ x = a$ or not, and, if the true identities include the likes of "Santa Claus = Santa Claus", then neither choice is entirely unobjectionable.

Quantified Modal Logic with an Existence Predicate

In defense of the Barcan formula and converse Barcan formula, it might be said that the problem is not that they are invalid, just that they are easily misinterpreted. If we read the quantifiers as saying "For everything that *exists*" and "There *exists* an *x* such that", then we will think that these formulas are invalid. But we can grant that the Barcan formula and the converse Barcan formula are valid if we read the quantifiers as "For everything *possible*" and "There is a *possible* thing *x* such that".

As for our counterexamples to the Barcan formula, if everything that is even *possible* necessarily has mass, then there could not be a possible world in which something does not have mass. There is a possible world in which I have a twin brother; and so there is some *possible* object of which we can say, "He might possibly have been my twin brother".

As for our counterexample to the converse Barcan formula, on the present reading of the quantifiers, the antecedent, $\Box\forall x\exists y\ x = y$, means that at each possible world, each possible thing is a possible thing, which is trivial. But likewise the consequent, $\forall x\Box\exists y\ x = y$, is trivial. It just says that for each possible thing, from the point of view of every possible world it is a possible thing.

But now we have a different problem. We do not want our quantifiers always to be understood as ranging over everything possible. For example, if I say, "No one ever lives to be more than 120 years old", that might be true. But it will not be true if it means "No *possible* person lives to be more than 120 years old". To address this difficulty, we can introduce an *existence* predicate E (a frontwards "e", as distinguished from the backwards "e" of the existential quantifiers). Then when we want to say that all *actual* things are *F*, without also saying that all *possible* things are *F*, we can write: $\forall x(Ex \rightarrow Fx)$. And when we want to say that some *actual* thing is *F*, we write $\exists x(Ex \wedge Fx)$.

All of our definitions of a modal structure, and satisfaction, remain just the same as in Simple Quantified Modal Logic. It's just that we now *think* of the interpretation of E as

assigning to E at world *w* the subset of objects in *D* that actually exist in *w*. For example, we have:

$\models_{SQML} \Box\exists x\ x = a$,

and even

$\models_{SQML} \exists x\Box x = a$,

but

$\not\models_{SQML} \exists x\Box(Ex \wedge x = a)$.

and

$\not\models_{SQML} \exists x(Ex \wedge \Box x = a)$.

This might be the best we can do within the framework of a bivalent semantics. If you think that neither "Pegasus flies" nor "Pegasus does not fly" should count as true, because Pegasus does not exist, then you will need to adopt a three-valued semantics—to allow that both of these sentences are neither true nor false.

Rigid designation

We have assumed that in each structure, the assignment of objects to individual constants is constant across worlds. That is, for all individual constants n, for all $w, w' \in W$, $\Sigma(n, w) = \Sigma(n, w')$.

We can have a different kind of denoting expression, called a *definite description,* of the form: (ιvG) (That's the Greek letter iota, followed by a variable, followed by a formula. The iota is supposed to be written upside down, but I don't think MS Word will let me do that.) We can think of this as a translation of "the *G*-thing". Definite descriptions, so written, form abbreviations. For any formula F,

$F(\iota vG)/u =_{def} \exists v(Ev \wedge G \wedge \forall x((Ex \wedge Gx/v) \rightarrow x = v) \wedge Fv/u)$.

For example, "The little green man walked in" means: "There is a *v* such that *v* is an existing little green man and every existing little green man is *v* and *v* walked in".

Definite descriptions are non-rigid designators because the things they "denote" will not be the same in each world.

The modal-logical difference between rigid designators and nonrigid designators is evident from the following:

$$a = b \models_{SQML} \Box\, a = b$$

$$(\iota xG) = (\iota yH) \not\models_{SQML} \Box\, (\iota xG) = (\iota yH)$$

The reason for the latter is that even though, the unique thing in *w* that is *G* may be the unique thing in *w* that is *H*, it does not follow that in every possible world *w′* the unique thing in *w′* that is *G* is the unique thing in *w′* that is *H*.

So, for example, while it may be true that Hesperus is Phosphorus and true that *necessarily* Hesperus is Phosphorus, and true that Hesperus is the first star seen in the evening and true that Phosphorus is the first star seen in the morning, and true that the first star seen in the evening is the first star seen in the morning, it is not true that *necessarily* the first star seen in the morning is the first star seen in the evening.

Counterpart Theory

There is at least one other idea from modal logic that frequently comes up in the philosophical literature, and that is the idea of *counterparts*. Some people, e.g., David Lewis, haven't liked the idea that a single individual can exist in different possible worlds, e.g., that I might belong to both the domain of objects in the actual world and belong to the domain of objects in some merely possible world. One thing they don't like about it is that it rules out the possibility that in some worlds there might be two people who have an equal claim on being me. And in some worlds there might be one person who has equal claim on being you *and* me. Suppose that you have an identical twin sister. But in some other world, your mother did not give birth to twins; she gave birth to just one daughter instead. In that other world, which person is the same person as you? Answer: that one daughter of your mother. But equally, that one daughter of your mother is the same person as your twin sister. (So *being the same person as* is not transitive.)

To arrange things so that our formal semantics reflects this idea, we may suppose that for any two worlds *w* and *w′*, the domain for *w* and the domain for *w′* are mutually exclusive. The extensions assigned to predicates at a world are formed exclusively from members of the domain for that world. We define *intensional objects* as *relations* whose members are

pairs consisting of a world and an object in the domain of that world (but the relation need not be a function). The objects assigned to individual constants and variables are such intensional objects. For example, if there are just three worlds, w_1, w_2, and w_3, and $o_1 \in \delta(w_1)$, $o_2 \in \delta(w_2)$, and $o_3 \in \delta(w_3)$, then an intensional object could be $\{\langle w_1, o_1\rangle, \langle w_2, o_2\rangle, \langle w_3, o_3\rangle\}$.

If we write a sentence using $\Box$ or $\Diamond$ and some individual constant n, then in order to decide whether it is true in world w, we have to consider the intensional object assigned to n. For example, suppose *Int* is the intensional object assigned to the individual constant n in some structure. Then $\Box\mathsf{Fn}$ will be true at w if and only if for all $o \in D$ and all $w' \in W$, if wRw' and $\langle w', o\rangle \in Int$, then $\langle o\rangle \in \Sigma(\mathsf{F}, w')$.